WSML Deliverable

D20 v0.1
OWL Lite$^-$

WSML Working Draft – July 18, 2004

Authors:
Jos de Bruijn
Axel Polleres
Dieter Fensel
Editors:
Jos de Bruijn
Reviewers:
Boris Motik

Final version:
`http://www.wsmo.org/2004/d20/v0.1/20040629/`
Latest version:
`http://www.wsmo.org/2004/d20/v0.1/`
Previous version:
`http://www.wsmo.org/2004/d20/v0.1/20040621/`

Abstract

This deliverable presents a restricted variant of the OWL Lite species of the OWL ontology language, called OWL Lite$^-$. OWL Lite$^-$ is a strict subset of OWL Lite and can be translated directly into the deductive database language Datalog. Thus, any OWL Lite$^-$ ontology can be translated into Datalog, in order to allow for efficient query answering. It turns out that most current ontologies fall inside this fragment.
An ontology language for which a translation to Datalog exists has several advantages. Most notably, it can benefit from highly optimized query answering engines, and allows for easy implementation of a rule and a query language on top of the ontology.
We describe the restrictions on the OWL Lite abstract syntax and provide an analysis of the features of OWL Lite, which are not included in OWL Lite$^-$, and give a rationale for not including them in the language. We present the RDF syntax and the model-theoretic semantics for OWL Lite$^-$ and relate OWL Lite$^-$ with RDFS. Finally, we indicate possible future extensions of the language.

Contents

1 Introduction **4**

2 Limitations of OWL Lite **7**

3 Preliminaries **10**
3.1 Description Logics 10
3.2 Logic Programming and Datalog 11
3.3 Description Logic Programs 12

4 OWL Lite⁻ **14**
4.1 From OWL Lite to First-Order Logic 14
4.2 Defining OWL Lite⁻ 16
4.3 OWL Lite⁻ RDF Syntax 22
4.4 OWL Lite⁻ Model-Theoretic Semantics 23
4.5 Transforming OWL Lite⁻ to Datalog 23
4.6 The relation between OWL Lite⁻ and RDFS 23
4.6.1 RDFS in OWL Lite⁻ 24
4.6.2 Restrictions on RDFS imposed by OWL Lite⁻ 25
4.6.3 Expressivity of OWL Lite⁻ compared to RDFS 25
4.7 Summary 26

5 Conclusions and Future Work **27**
5.1 Datatypes in OWL Lite⁻ 27
5.1.1 Datatypes in OWL 27
5.1.2 Ways of handling datatypes in OWL Lite⁻ 28
5.2 Possible Future extensions of OWL Lite⁻ 28

1 Introduction

The Web Ontology Language OWL [Dean and Schreiber, 2004] consists of three species, namely OWL Lite, OWL DL and OWL Full. The OWL Lite and OWL DL species are syntactical variants of Description Logic languages, namely, OWL Lite and can be seen as a variant of the $\mathcal{SHIF}(\mathbf{D})$ description logic language, whereas OWL DL is a variant of the $\mathcal{SHOIN}(\mathbf{D})$ language [Horrocks and Patel-Schneider, 2003]. The $\mathcal{SHIF}(\mathbf{D})$ language allows complex class descriptions, including conjunction, disjunction, negation, existential and universal value restrictions, role hierarchies, transitive roles, inverse roles, a restricted form of cardinality constraints (cardinality 1) and support for concrete domains. $\mathcal{SHOIN}(\mathbf{D})$ adds support for individuals in class descriptions and arbitrary cardinality constraints. In other words, OWL DL is as extension of OWL Lite, adding support for individual names in class descriptions (also called *nominals*), and allowing arbitrary cardinality restrictions[1]. OWL Full is a syntactic and semantic extension of both OWL DL and RDF(S) and thus cannot be translated into a Description Logic language. Entailment in OWL Full is undecidable in the general case, because it allows arbitrary roles in number restrictions, which makes the logic undecidable [Horrocks et al., 2000]; it is expected, but not proven, that entailment in OWL Full can be reduced to satisfiability in a First-Order Logic[2].

Reasoning in the $\mathcal{SHIF}$[3] language (checking satisfiability of the knowledge base) is ExpTime-complete. Furthermore, ABox reasoning (more specifically, checking class membership and instance retrieval) in most Description Logic reasoners is very expensive, because for each individual the negation of the membership needs to be added to the Knowledge Base and satisfiability checking is required. When retrieving all members of a class, this satisfiability checking has to be done for each individual in the knowledge base. This is clearly not scalable in the case of large knowledge bases with many individuals. There are several attempts at increasing the efficiency of instance retrieval. Most of these attempts rely on a form of caching of query results and/or class-membership relationships (also called realization of the ABox). The RACER Description Logic reasoner implements several optimization techniques for ABox reasoning, including caching query results, realization of the ABox and dynamic indexing (see [Haarslev and Möller, 2003; Haarslev and Möller, 2004] for more detailed descriptions).

Another attempt at increasing the efficiency of ABox reasoning is the instance store [Horrocks et al., 2004], developed at the University of Manchester. In the instance store approach, Abox reasoning in Description Logics is (partly) optimizied by storing information about individuals in a relational database. Horrocks et al. optimize instance retrieval by storing individual assertions in a database and by caching the classification of descriptions in the ontology. All assertions of the form $i \in D$, where i is an individual and D is a description, are stored in the database, along with the complete classification of all descriptions in the ontology, which include the equivalence and subsumption relationships for each description. Most instance retrieval queries can now be

[1] All other features of OWL DL can be encoded in OWL Lite by introducing new class names for complex descriptions [Horrocks et al., 2003] and by introducing range restrictions using the top ($\top$) and bottom ($\bot$) concepts for negation [Volz, 2004, pp. 63].

[2] Based on a discussion thread on the www-rdf-logic mailing list: http://lists.w3.org/Archives/Public/www-rdf-logic/2004May/0054.html

[3] For now we disregard concrete domains in our treatment of OWL Lite. This will be part of future work, see also section 5.1.

reduced to database querying. The Description Logic reasoner is only required when retrieving instances of a complex description, when there is no equivalence between this complex description, with a primitive concept stored in the database, therefore, the worst-case complexity of instance retrieval for arbitrary descriptions remains the same. However, for primitive concepts the complexity is significantly reduced. The major limitations of this instance store are: (1) no relations between instances (i.e. property fillers) can be stored in the database, although there are some ideas on how to support property fillers in a limited way and (2) if the ontology is updated, the classification needs to be recomputed and, most likely, the instance store in the database needs to be recreated.

Another way to optimize ABox reasoning is by using Logic Programming (or Deductive Database) formalisms (e.g. [Grosof et al., 2003; Motik et al., 2003]) for query answering. By using deductive databases, it is possible to benefit from the many years of research in optimizing query answering.

In this deliverables, we will take the DLP (Description Logic Programs) approach of [Grosof et al., 2003], which was further elaborated by Raphael Volz in his PhD dissertation [Volz, 2004], as a starting point for our analysis of OWL Lite and our development of the OWL Lite$^-$ language. DLP uses a subset of the Description Logic language $\mathcal{SHOIN}$ (which underlies OWL DL), that can be directly translated into the deductive database language Datalog, in order to perform efficient ABox reasoning.

Limitations in ABox reasoning are not the only possible drawback we see in the use of OWL as an ontology language for the Semantic Web. We feel that the Lite species of OWL is still too heavy, in terms of both expressivity and complexity, for many uses on the Semantic Web. Features such as cardinality restrictions, which introduce equality, behave in a manner which is non-intuitive for many users. There are also problems with rule extensions of Description Logic languages. The combination of Description Logics with (function-free) Horn Logic easily leads to undecidability, when combined in a naive way, as was done in the proposal for a Semantic Web rule language SWRL [Horrocks and Patel-Schneider, 2004], which was recently submitted to the W3C[4].

We propose to use a subset of OWL Lite, called OWL Lite$^-$ as a starting point for a Semantic Web ontology language, based on the $\mathcal{L}_0$ language identified by Raphael Volz in his PhD dissertation [Volz, 2004]. The $\mathcal{L}_0$ language is roughly speaking the maximal subset of Description Logics and Datalog[5], and thus allows for easy extensibility in both the terminological and the rule direction.

There are various benefits of choosing $\mathcal{L}_0$ as a starting point. First of all, $\mathcal{L}_0$ can be directly rewritten into the Deductive Database language Datalog, which is data complete for the complexity class P [Dantsin et al., 2001], meaning that query answering can be done in polynomial time with respect to the size of the database. Furthermore, there are many existing efficient implementations of (a superset of) Datalog, such as DLV[6], XSB[7], KAON[8], SWI-Prolog[9] and OntoBroker[10]. Second, $\mathcal{L}_0$ can be extended in a straightforward manner in both the Description Logic (DL) world, and the Logic Programming (LP) (or Deductive Database) world. In the DL world, $\mathcal{L}_0$ can be extended to OWL DL,

[4] http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/

[5] Recent research may have shown that a larger Description Logic fragment can be expressed in Datalog, which would invalidate this claim. Future versions of this deliverable will include an analysis of the Horn subset of $\mathcal{SHIQ}(\mathbf{D})$, which was identified by [Hustadt et al., 2004].

[6] http://www.dbai.tuwien.ac.at/proj/dlv/

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

[8] http://kaon.sourceforge.net/

[9] http://www.swi-prolog.org/

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

in order to make use of the full feature set which Description Logics has to offer, while maintaining decidability. In the LP world, $\mathcal{L}_0$ can be extended with, for example, rules and non-monotonic features such as default negation. We can also envision more advanced extensions based on languages such as HiLog [Chen et al., 1993] and F-Logic [Kifer et al., 1995].

In the following pages, we first identify the major limitations of OWL Lite in chapter 2. In chapter 3 we briefly review Description Logics, Logic Programming and Description Logic Programs. In chapter 4 we describe the actual OWL Lite$^-$ language. We end up with conclusions and future work in chapter 5.

2 Limitations of OWL Lite

OWL Lite is the least expressive species of OWL. However, this language already requires reasoning with equality, which significantly increases computational complexity. Cardinality restrictions, in their current form, introduce equality in a non-intuitive way, as explained below. There is no notion of constraints in OWL Lite. Furthermore, because the expressiveness of the $\mathcal{SHIF}$ Description Logic language is beyond the capabilities of efficient rule-based engines, and because straightforwardly extending a Description Logic with Horn-like rules leads to undecidability issues [Levy and Rousset, 1998], one cannot easily extend OWL Lite with a rule language.

ABox Reasoning in OWL Lite is hard Reasoning with OWL Lite requires reasoning with equality, which is hard to do. The satisfiability problem in $\mathcal{SHIF}$ has ExpTime complexity[1]. In our opinion it is a mistake to require such complex reasoning for the least expressive of the OWL species, since efficient ABox reasoning should play a major role in this fragment.

As was pointed out in the introduction, ABox reasoning in a Description Logic knowledge base is very expensive, because each individual to be retrieved requires a check on unsatisfiability of the knowledge base[2]. There are currently over four billion web pages indexed by Google, therefore, in order for the Semantic Web to work outside of the research lab, it must be possible to reason with large collections of instances.

Deriving Equality in OWL Lite is non-intuitive In OWL Lite, it is possible to specify functional properties (properties with a cardinality of at most one). However, when two instances of this property have the same domain value, but a different range value, the two instances in the range will be deemed equal according to the semantics of OWL Lite. We illustrate this with an example:

```
ObjectProperty(hasSpouse domain(Person) range(Person))
Class(Person restriction(hasSpouse maxCardinality(1)))

Individual(john type(Person) value(hasSpouse mary)
   value(hasSpouse jennifer))
```

From the above example the reasoner will draw the conclusion that `mary` and `jennifer` both refer to the same person. The possibilities, that there was a modeling mistake (either at the ontology or at the instance level), or that `john` is breaking the law (assuming bigamy is illegal), are not taken into account[3]. This is highly related to the next limitation of OWL Lite (and OWL in general): the lack of constraints.

[1] To be fair, we must note here that the ExpTime complexity of $\mathcal{SHIF}$ refers to the combined complexity, and the P complexity of Datalog refers to the data complexity. In fact, the combined complexity of Datalog is also ExpTime. However, in our opinion, not only the theoretical (worst-case) complexity counts, but also the difficulty in providing full reasoning support and the availability of actual (optimized) implementations.

[2] We must note that there exist several approaches for more efficient ABox reasoning (e.g. [Horrocks et al., 2004; Grosof et al., 2003; Haarslev and Möller, 2004]), however, there are currently no optimized approaches which capture full ABox reasoning in OWL Lite.

[3] In fact, the authors of F-Logic [Kifer et al., 1995] uncovered the same problem with their language and proposed to solve it by allowing the user to specify which part of the program is allowed to infer equality and which part of the program would infer a modeling mistake [Kifer et al., 1995, Section 12.1].

Lack of a notion of constraints in OWL Lite OWL Lite lacks a notion of constraints. One would intuitively expect that (value and cardinality) restrictions in OWL Lite correspond to constraints, which is not the case. Value restrictions are used to *infer* new information about individuals, rather than to detect inconsistencies. We will give an example of this below. Cardinality restrictions are used to derive equality, as was shown in the description of the previous limitation.

In order to illustrate how value restrictions infer new information about individuals, we use the class and property descriptions from the previous example (i.e. the class `Person` and the property `hasSpouse`) and add one description and two assertions:

```
Class(Cow)

Individual(belle type(Cow))
Individual(mark type(Person) value(hasSpouse belle))
```

From the above assertions we can conclude that there is a cow `belle` and that there is a person `mark`; `mark` has as a spouse `belle` and because `belle` is in the range of the property `hasSpouse`, `belle` *is a* `person`. We argue that the last conclusion is counter-intuitive. We know that a cow is not a person and vice versa. The only way to model value constraints, in the sense of database constraints, in OWL Lite is by explicitly asserting the disjointness of the two classes[4] or by asserting inequality for each distinct pair of individuals.

Note that the lack of a notion of constraints in Description Logics was acknowledged in the Description Logic community (e.g. [Donini et al., 1998b; Donini et al., 2002]). [Donini et al., 1998a] introduces the epistemic operator **K**. When using the **K** operator as a prefix to a concept (**K**C) or role (**K**R) description, the description intuitively represents all the knowledge in the knowledge base about the concept or role (as opposed to the usual Description Logic descriptions, which represent all knowledge in the world), i.e. all the individuals *known* to be instances of C and all property fillers *known* to be instances of R. In a sense, the **K** operator is used to capture the notion of minimal knowledge, a common notion in logic programming. Epistemic queries can be used to formulate integrity constraints. An answer to the query is then only generated, if the instance violating the integrity constraint is in the knowledge base. This corresponds with the usual notion of integrity constraints in deductive databases.

Rule extension of OWL Lite is not straightforward There have been several proposals for rule extensions of Description Logic languages. Two general directions can be identified in these approaches: (1) approaches which allow for the use of classes and roles from the Description Logic knowledge base to occur as unary and binary predicates, respectively, in Horn clauses (e.g. $\mathcal{AL}$-log [Donini et al., 1998b] and CARIN [Levy and Rousset, 1998]) and (2) approaches which directly extend the Description Logic knowledge base with Horn-style rules (e.g. SWRL [Horrocks and Patel-Schneider, 2004] and [Motik et al., 2004]). The main distinction between the two kinds of approaches is the fact that $\mathcal{AL}$-Log and CARIN do not allow predicates from the Description Logic knowledge base in the heads of the rules, whereas the latter two approaches do, which means that in these approaches conclusions drawn from the rules affect the knowledge base.

All approaches mentioned above require Description Logic reasoning for evaluation of the knowledge base, except for the approach presented in [Motik et al.,

[4] Disjointness can not be directly modeled in OWL Lite. However, one can create a complete class definition with the disjoint classes on the right-hand side and `owl:Nothing` on the left-hand side, e.g. `Class(owl:Nothing complete Person Cow)`, which is equivalent to the following Description Logic statement: $\bot \equiv Person \sqcap Cow$.

2004]. Motik et al. translate the description logic knowledge base into a disjunctive datalog program. So-called DL-safe rules can be appended to this program, which allows for integrated reasoning with both the knowledge base and the rules. However, adding disjunction to a logic program significantly increases complexity. In fact, disjunctive datalog is data complete (for positive queries) for co-NP [Dantsin et al., 2001] (i.e. query answering has a worst-case complexity of co-NP). Note that in the approach of Motik et al., when not using disjunction, there is graceful degradation, i.e. query answering is in P if there is no disjunction in the head.

One note about SWRL is in order here. Satisfiability of an OWL DL knowledge base augmented with SWRL rules is undecidable, as was pointed out by the authors of the proposal [Horrocks and Patel-Schneider, 2004]. This undecidability is shown by the fact that SWRL rules can be used to simulate role value maps. Similarly, it has been shown that such straightforward combination of Description Logic with Horn rules, can be used to simulate Turing machines [Levy and Rousset, 1998], which clearly leads to undecidability.

We argue that some of the above mentioned limitations can be overcome by using a more restricted form of OWL Lite, which can be translated into a Datalog program (without equality). This language can then be straightforwardly extended to include database-style integrity constraints, which can be used for both cardinality and value constraints. Furthermore, in Datalog rules can be added directly on top of the ontology.

3 Preliminaries

In order to make this paper self-contained, we will first describe several preliminaries, necessary to explain and motivate our restriction of OWL Lite. We will first give a short introduction to Description Logics and Datalog. We will then describe the fragment of Description Logics, which can be expressed in Datalog, called *Description Logic Programs* (DLP), which was described by Raphael Volz in his PhD dissertation [Volz, 2004].

3.1 Description Logics

Description Logics [Baader et al., 2003] (formerly called *Terminological Logics*) are a well-known family of knowledge representation languages, which revolve mainly around concepts, roles, which denote relationships between concepts, and role restrictions. Concepts can be seen as unary predicates, whereas roles can be seen as binary predicates, although there are also Description Logic languages which allow n-ary roles[1], e.g. $\mathcal{DLR}$ [Calvanese et al., 1998].

Besides the concepts and role descriptions, which comprise the so-called TBox, a Description Logic Knowledge base typically also contains individuals (instances) and relations between individuals and, in the case of OWL, equality and inequality assertions between individuals. These assertions about individuals comprise the so-called ABox. The assertions in the ABox are of the form $i \in D$, where i is an individual and D is a description, and of the form $\langle i_1, i_2 \rangle \in R$, where i_1 and i_2 are individuals and R is a (binary) role.

Description Logic languages typically have set-based model-theoretic semantics, in which a description D is mapped to a subset of a domain $\Delta^{\mathcal{I}}$ under an interpretation $\mathcal{I}$ using a mapping function $\cdot^{\mathcal{I}}$. Similarly, a role R is mapped to a binary relation over the domain $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Equivalence of descriptions is interpreted as equal subsets ($C \equiv D$ is interpreted as $C^{\mathcal{I}} = D^{\mathcal{I}}$), subsumption is interpreted as a subset relation ($C \sqsubseteq D$ is interpreted as $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$), and so on. We refer the interested reader to [Baader et al., 2003, Chapter 2] for a more exhaustive treatment of Description Logic semantics.

An important aspect of Description Logic languages such as $\mathcal{SHIF}$ and $\mathcal{SHOIN}$ (the languages underlying OWL Lite and OWL DL, respectively) is that they form a decidable subset of First-Order Logic, i.e. given any knowledge base (i.e. TBox and ABox) in a decidable Description Logic, it is possible to decide in finite time whether the formula is satisfiable.

Current Description Logic reasoners, such as FaCT and Racer, have optimized algorithms for TBox reasoning. Typical reasoning tasks for TBox reasoning include subsumption checking, i.e. checking whether one concept is subsumed by another and satisfiability checking, i.e. checking whether a model exists in which the description is not mapped to the empty set ($C^{\mathcal{I}} \neq \emptyset$). Subsumption checking can be reduced to satisfiability and vice versa.

Although ABox reasoning is not supported by all Description Logic reasoners, there are important reasoning tasks to be performed in the ABox. Typical ABox reasoning problems include instance retrieval, i.e. retrieving all instances of a particular description, and property filler retrieval, i.e. retrieving all in-

[1] Because most popular Description Logic languages only allow binary roles, including the languages underlying the current Web Ontology Language OWL, we will restrict ourselves to binary roles in our treatment of Description Logics

stances which are related to a particular instance a via a particular property R. Typically, ABox reasoning is reduced to TBox reasoning. To check whether a particular instance i is in a particular description D, the assertion $\neg i \in D$ is added to the knowledge base and the knowledge base is checked for unsatisfiability. If the knowledge base is unsatisfiable, it means that $i \in D$. Clearly, this type of ABox reasoning is very expensive for large ABoxes, however, there does exist a solution to optimize part of the ABox reasoning problem using a relational database, called *instance store* [Bechhofer et al., 2002]. The instance store was already described in detail in the introduction.

3.2 Logic Programming and Datalog

Classical Logic Programming makes use of the Horn logic fragment of First-Order Logic. A First-Order formula is in the Horn fragment, if it is a disjunction of literals with at most one positive literal, in which all variables are universally quantified:

$$p_1 \vee \neg n_1 \vee \ldots \vee \neg n_k \tag{3.1}$$

This formula can be rewritten in the following form:

$$p_1 \leftarrow n_1 \wedge \ldots \wedge n_k \tag{3.2}$$

Such a formula is also called a *Horn formula*. A Horn formula with one positive literal, and at least one negative literal is called a *rule*. The positive literal p_1 is called the *head* of the rule. The conjunction of negative literals $n_1 \wedge \ldots \wedge n_n$ is called the *body* of the rule. A rule without a body is called a *fact* and a rule without a head is called a *query*.

There are two basic styles for defining the semantics of logic programs. The first is the model-theoretic semantics, in which the semantics of a program P is given by the minimal Herbrand model M_P. The other style of semantics is the computational semantics, in which the semantics of a program P is given by the least fixpoint of the direct-consequence operator T_P. A single application of T_P yields all atoms that can be derived by a single application of some rule in P given the atoms in the interpretation I

T_P has a least fixpoint T_P^∞. This fixpoint is reached if no atoms can be derived from the application of T_P that are not in I. An important result in Logic Programming is that the least fixpoint of T_P coincides with the minimal Herbrand model M_P. For a more exhaustive treatment of the syntax and semantics of logic programming, we refer the reader to [Lloyd, 1987].

Interest in the use of logic for database has given rise to the field of *deductive databases*. Datalog is the most prominent language for deductive databases. A datalog program corresponds to a logic program with the following restrictions: datalog allows only *safe rules*, i.e. every variable occurring in the head of a rule must also occur in the body of the rule and datalog *disallows the use of function symbols*.

In datalog, the (deductive) database consists of two parts, namely the extensional database (EDB), consisting of a set of facts, and the intensional database (IDB) consisting of a set of rules. The predicates occurring in EDB are called *extensional predicates* and the predicates occurring in the heads of the rules in IDB are called *intensional predicates*. We assume, without loss of generality, that the sets of extensional and intensional predicates are disjoint, which means that no extensional predicate is allowed to occur in the head of a rule in IDB. For more information on Datalog, see [Ullman, 1988].

The most prominent reasoning task, in both Logic Programming and Deductive Databases, is *query answering*. A query can either be a *ground atom query*, where the task is to determine whether a ground atom A is entailed by the program or an *open atom query*, where the task is to retrieve all the variable substitutions for an atom A. Open atom queries can actually be reduced to ground atom queries.

The complexity of query answering is usually divided into data complexity and program complexity. The *data complexity* is the complexity of checking whether $EDB \bigcup P \models A$ where the logic program P is *fixed* and the extensional database EDB and the ground atoms A are an *input*. The *program complexity* is the complexity of checking whether $EDB \bigcup P \models A$ where the extensional database EDB is *fixed* and the logic program P and the ground atoms A are an *input*. It turns out that Datalog is data complete for P, i.e. query answering with a fixed program P has a worst-case polynomial complexity, and program complete for $\mathsf{ExpTime}$.

Interestingly, for a certain class of queries, namely *unbounded queries*[2], the expressive power of Datalog goes beyond the expressive power of First-Order Logic[Ajtai and Gurevitch, 2001].

Many well-known extensions for Datalog exist, such as different forms of negation and the use of function symbols. Although unrestricted use of function symbols makes the program undecidable, many well-known restrictions on the use of function symbols exist, which are known to be decidable (e.g. [Bonatti, 2004]).

3.3 Description Logic Programs

[Volz, 2004] defines the DLP (Description Logic Programming) language $\mathcal{L}_0$ as the intersection of the expressive description logic $\mathcal{SHOIN}$ (the language underlying OWL DL) and Datalog.

$\mathcal{L}_0$ was originally developed to optimize reasoning with individuals on the Semantic Web. $\mathcal{L}_0$ places several restrictions on the use of Description Logic constructs, because the intersection of Description Logics and Datalog is in many respects less expressive than Description Logics, e.g. it does not allow arbitrary negation, disjunction in the head or existential quantification. One important divergence from Description Logics is that it distinguishes between constructs allowed on the left-hand side and the right-hand side of the inclusion symbol ($\sqsubseteq$). Cardinality restrictions and individual (in)equality assertions are not allowed, because equality is not in Datalog. An example for this distinction between the left- and the right-hand side is disjunction, which can be expressed on the left-hand side, but not on the right-hand side of the inclusion symbol.

$\mathcal{L}_0$ has the following properties, which make it a good candidate to be the basis for an ontology language on the Semantic Web:

- $\mathcal{L}_0$ contains the most frequently occurring Description Logic modeling primitives. It turns out that most OWL (and DAML+OIL) ontologies in popular ontology libraries fall in this fragment [Volz, 2004].
- The most important reasoning task in deductive databases is query answering. There has been significant research on the properties of Datalog with respect to the task of query answering. In $\mathcal{L}_0$, all individual-related

[2]Unbounded queries are queries that cannot be transformed to non-recursive queries, i.e. queries with no intensional predicates in the rule bodies.

reasoning tasks can be reduced to query answering in Datalog. This allows us to benefit from the research and implementations in the deductive database area.

- Because $\mathcal{L}_0$ is a proper subset of both Datalog and a Description Logic language, it is straightforward to extend the language in either direction. In fact, $\mathcal{L}_0$ can be used to allow inter-operation between the two paradigms.

Our reason for choosing $\mathcal{L}_0$ as the basis for the OWL Lite$^-$ language is three-fold:

- There exist many efficient implementations of Datalog, which are very efficient at the task of query answering.
- It turns out that most ontologies currently on the Semantic Web can be expressed in the language $\mathcal{L}_0$ [Volz, 2004].
- It turns out that nearly all of $\mathcal{L}_0$ is included in OWL Lite. In fact, it turns out that the only construct which is in $\mathcal{L}_0$, but is not in OWL Lite, is the `hasValue` property restriction ($\exists R.\{o\}$).

4 OWL Lite⁻

This chapter defines the ontology language OWL Lite⁻, which is a proper subset of OWL Lite that can be translated into Datalog.

In this chapter, we restrict the syntax and semantics of OWL Lite to create OWL Lite⁻ and provide a discussion of all the features of OWL Lite, which are not in OWL Lite⁻.

The chapter is structured as follows. First, we provide a satisfiability-preserving translation from OWL Lite to First-Order Logic. Then we define OWL Lite⁻ and enumerate all the feature which are in OWL Lite⁻ and all the features which have been omitted. We provide the restrictions on the abstract syntax and the RDF syntax of OWL Lite, and we restrict the model-theoretic semantics of OWL Lite to those of OWL Lite⁻. We then provide a direct translation from OWL Lite⁻ to Datalog. Finally, we describe the relationship between RDFS and OWL Lite⁻.

4.1 From OWL Lite to First-Order Logic

This section summarizes the constructs in OWL Lite and provides a translation to Description Logic (DL) and First-Order Logic (FOL) with equality. The mapping between OWL Lite and Description Logic is partly based on [Horrocks et al., 2003]; for the mapping from Description Logic to First-Order Logic we use the mapping function π presented in [Borgida, 1996].

Please note that we use the OWL abstract syntax [Patel-Schneider et al., 2004] for writing down OWL Lite statements as opposed to the RDF/XML syntax, because of its readability and understandability.

We have split the mappings into two tables, following the approach in [Horrocks et al., 2003]. The OWL Lite Descriptions are presented in Table 4.1; the axioms and facts are presented in Table 4.2. Note that in Table 4.2, C stands for a named class and D stands for either a named class or a property restriction.

OWL Abstract Syntax	**DL syntax**	**FOL syntax**
Descriptions (C)		
A (URI Reference)	A	$A(x)$
`owl:Thing`	$\top$	$true$
`owl:Nothing`	$\bot$	$false$
`restriction(`R `someValuesFrom(`C`))`	$\exists R.C$	$\exists y : (R(x,y) \wedge C(y))$
`restriction(`R `allValuesFrom(`C`))`	$\forall R.C$	$\forall y : (R(x,y) \rightarrow C(y))$
`restriction(`R `minCardinality(0))`	$\geqslant 0R$	$true$
`restriction(`R `minCardinality(1))`	$\geqslant 1R$	$\exists y : R(x,y)$
`restriction(`R `maxCardinality(0))`	$\leqslant 0R$	$\forall y : \neg R(x,y)$
`restriction(`R `maxCardinality(1))`	$\leqslant 1R$	$\forall y_1, y_2 : R(x,y_1) \wedge R(x,y_2) \rightarrow y_1 = y_2$

Table 4.1: Mapping of OWL Lite Descriptions to DL and FOL

A few things must be noted about the translation from OWL Lite to FOL:

- For the translation from DL into FOL we used the mapping function π as presented in [Borgida, 1996]. In many cases, the OWL Lite axiom is not an elementary axiom, so that the translation of the Description Logic axiom into FOL requires several applications of the mapping function π.

OWL Abstract Syntax	DL syntax	FOL syntax
`Class(`C `partial` C_1 `...` C_n`)`	$C \sqsubseteq D_1 \sqcap \ldots \sqcap D_n$	$\forall x : (C(x) \rightarrow \bigwedge \pi(D_i, x)$
`Class(`C `complete` C_1 `...` C_n`)`	$C \equiv D_1 \sqcap \ldots \sqcap D_n$	$\forall x : (C(x) \leftrightarrow \bigwedge \pi(D_i, x))$
`EquivalentClasses(`C_1 `...` C_n`)`	$C_1 \equiv \ldots \equiv C_n$	$\forall x : (C_1(x) \leftrightarrow \ldots \leftrightarrow \pi(C_n, x))$
`ObjectProperty(`R `super(`R_1`) ... super(`R_n`)`	$R \sqsubseteq R_i$	$\forall x, y : R(x, y) \rightarrow \bigwedge R_i(x, y)$
`domain(`C_1`) ... domain(`C_n`)`	$\top \sqsubseteq \forall U^-.C_i$	$\forall x, y : R(x, y) \rightarrow \bigwedge C_i(x)$
`range(`C_1`) ... range(`C_n`)`	$\top \sqsubseteq \forall R.C_i$	$\forall x, y : R(x, y) \rightarrow \bigwedge C_i(y)$
`[inverseOf(`R_0`)]`	$R \equiv ({}^-R_0)$	$\forall x, y : R(x, y) \leftrightarrow R_0(y, x)$
`[Symmetric]`	$R \equiv ({}^-R)$	$\forall x, y : R(x, y) \leftrightarrow R(y, x)$
`[Functional]`	$\top \sqsubseteq \leqslant 1R$	$\forall x, y_1, y_2 : R(x, y_1) \wedge R(x, y_2) \rightarrow y_1 = y_2$
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1R^-$	$\forall x_1, x_2, y : R(x_1, y) \wedge R(x_2, y) \rightarrow x_1 = x_2$
`[Transitive])`	$\mathsf{Trans}(R)$	$\forall x, y, z : R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$\forall x, y : (R_1(x, y) \rightarrow R_2(x, y))$
`EquivalentProperties(`R_1 `...` R_n`)`	$R_1 \equiv \ldots \equiv R_n$	$\forall x, y : (R_1(x, y) \leftrightarrow \ldots \leftrightarrow R_n(x, y))$
`Individual(`o `type(`C_1`) ... type(`C_n`)`	$o \in C_i$	$C_i(o)$
`value(`R_1 o_1`) ... value(`R_n o_n`))`	$\langle o, o_i \rangle \in R_i$	$R_i(o, o_i))$
`SameIndividual(`o_1 `...` o_n`)`	$o_1 = \ldots = o_n$	$o_1 = \ldots = o_n$
`DifferentIndividuals(`o_1 `...` o_n`)`	$o_i \neq o_j, i \neq j$	$o_i \neq o_j, i \neq j$

Table 4.2: Mapping of OWL Lite Axioms and Facts to DL and FOL

- The mapping function π appears twice in the translated FOL sentences, namely in the first and second line of Table 4.2. This is because classes can be defined by *named classes* and/or *restrictions*. Occurrences of D_i correspond with either named classes or property restrictions. All occurrences of C_i and R_i in the translated sentences correspond to named classes and named properties, respectively.

- `owl:Thing` ($\top$) and `owl:Nothing` ($\bot$) are translated to *true* and $false$, respectively. By doing this we follow the approach in [Hustadt et al., 2004], which differs slightly from the original approach from Borgida [Borgida, 1996]. Borgida introduces an equality $x = x$ for $\top$ and an inequality $x \neq x$ for $\bot$. The interpretation of *true* is equivalent to $x = x$ and the interpretation of $false$ is equivalent to $x \neq x$, however, by introducing the special symbols *true* and $false$, equality is not necessary in the language.

  Note that in many cases the *true* and $false$ symbols can be eliminated after the translation into FOL. For example, when translating $\geqslant 0R$ into FOL, we obtain *true*. If this restriction occurs in a class definition with other restrictions or named classes, it can be eliminated.

- For the translation from DL into FOL we use the preprocessing steps presented in [Volz, 2004, Table 4.1 and 4.2, pp. 84,85]. These normalization and simplification steps help in many cases to simplify the translation into FOL. As an example, we show the translation of the OWL Lite description `restriction(`R `maxCardinality(0))` into FOL.

  Example 1. *According to [Horrocks et al., 2003],*
  `restriction(`R `maxCardinality(0))`
  is equivalent to the Description Logic statement $\leqslant 0R$. According to the previously mentioned preprocessing rules, this is equivalent to the Description Logic statement $\forall R.\bot$. When applying the translation function π

twice, we obtain the FOL sentence $\forall x, y : R(x, y) \rightarrow false$*. According to De Morgan, this can be rewritten to:* $\forall x, y : \neg R(x, y) \vee false$*. Any occurrence of* $false$ *in a disjunction can be eliminated, yielding* $\forall x, y : \neg R(x, y)$*.*

The translations presented in Tables 4.1 and 4.2 provide the translation from OWL Lite into First-Order Logic. The next step is the transformation of the First-Order Logic formulae into Horn formulae. If such a transformation exists, it means that the feature is in OWL Lite⁻. The next section identifies exactly which of these features can be translated into Horn formulas and are thus in OWL Lite⁻.

4.2 Defining OWL Lite⁻

Table 4.3 presents all the OWL Lite primitives and identifies for each primitive whether it can be expressed in $\mathcal{L}_0$ and thus is part of OWL Lite⁻. This table is partly based on [Volz, 2004, Table 4.7, pp. 111].

First of all, we need to note here that the classification of features in OWL Lite that are in OWL Lite⁻ differs slightly from the approach taken by Volz [Volz, 2004] in describing the $\mathcal{SHOIN}$ primitives included in different logic programming variants. We do not explicitly distinguish between the right-hand side and the left-hand side of the general class inclusion GCI ($\sqsubseteq$). We follow the OWL Lite abstract syntax, which distinguishes between partial and complete class definitions. Therefore, any statement that is allowed to occur on the right-hand side is allowed to occur in the defining part of a partial class definition (note that in OWL Lite, only named classes are allowed to occur in the defined part of a class definition) and any statement that is allowed to occur on both the left-hand side and the right-hand side is allowed to occur in the defining part of a complete class definition. Note that $\mathcal{L}_0$ allows disjunction and existential property restrictions on the left-hand side of the GCI. A disjunction on the left-hand side can be encoded as several partial class definitions with the same defining part (the right-hand side), where the defined class coincides with one of the classes of the disjunction. An existential property restriction can not occur just on the left-hand side of the GCI using the OWL Lite abstract syntax, therefore, because the OWL Lite⁻ abstract syntax is based on the OWL Lite abstract syntax, the existential property restriction is not allowed in OWL Lite⁻.

We will now discuss for each feature in OWL Lite, where a feature corresponds to a line in Table 4.3, why it is or is not included and we will discuss what is the cause of the fact that the feature could not be included and what would be the implications for the complexity of the language if the feature were to be included.

For our discussion about the complexity of the features in OWL Lite we will use four Logic Programming variants. These four variants have increasing complexity and are strictly semantically layered, i.e. each variant with a higher number is strictly more expressive than the lower variants. The variants we distinguish are:

- LP_0: Datalog, which is the language used as a basis for OWL Lite⁻. Each feature of OWL Lite expressible in LP_0 is in OWL Lite⁻.
- LP_1: Datalog(IC), which generalizes LP_0 by allowing the use of integrity constraints. An integrity constraint can be seen as a Horn formula with an empty head. If the body of the rule is satisfied, the constraint is violated

OWL Abstract Syntax	**DL syntax**	**OWL Lite⁻**
`Class(`C `partial` C_1 ... C_n`)`	$C \sqsubseteq C_i$	$+$ $(C_i \neq \bot; C \neq \top)$
`Class(`C `complete` C_1 ... C_n`)`	$C \equiv C_1 \sqcap \ldots \sqcap C_n$	$+$ $(C, C_i \neq \bot, \top)$
`EquivalentClasses(`C_1 ... C_n`)`	$C_1 \equiv \ldots \equiv C_n$	$+$ $(C, C_i \neq \bot, \top)$
`ObjectProperty(`R `super(`R_1`)...super(`R_n`)`	$R \sqsubseteq R_i$	$+$
`domain(`C_1`)` ... `domain(`C_n`)`	$\top \sqsubseteq \forall R^-.C_i$	$+$ $(C_i \neq \bot)$
`range(`C_1`)` ... `range(`C_n`)`	$\top \sqsubseteq \forall R.C_i$	$+$ $(C_i \neq \bot)$
`[inverseOf(`R_0`)]`	$R \equiv ({}^-R_0)$	$+$
`[Symmetric(`R_0`)]`	$R \equiv ({}^-R)$	$+$
`[Functional]`	$\top \sqsubseteq \leqslant 1R$	–
`[InverseFunctional]`	$\top \sqsubseteq \leqslant 1R^-$	–
`[Transitive])`	$\mathsf{Trans}(R)$	$+$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$+$
`EquivalentProperties(`R_1 ... R_n`)`	$R_1 \equiv \ldots \equiv R_n$	$+$
`Individual(`o `type(`C_1`)` ... `type(`C_n`)`	$o \in C_i$	$+$
`value(`R_1 o_1`)` ... `value(`R_n o_n`))`	$\langle o, o_i \rangle \in R_i$	$+$
`SameIndividual(`o_1 ... o_n`)`	$o_1 = \ldots = o_n$	–
`DifferentIndividuals(`o_1 ... o_n`)`	$o_i \neq o_j, i \neq j$	–
Descriptions (C)		
A (URI Reference)	A	$+$
`owl:Thing`	$\top$	–
`owl:Nothing`	$\bot$	–
`restriction(`R `someValuesFrom(`C`))`	$\exists R.C$	–
`restriction(`R `allValuesFrom(`C`))`	$\forall R.C$	partial* $(C \neq \bot)$
`restriction(`R `minCardinality(0))`	$\geqslant 0R$	partial
`restriction(`R `minCardinality(1))`	$\geqslant 1R$	–
`restriction(`R `maxCardinality(0))`	$\leqslant 0R$	–
`restriction(`R `maxCardinality(1))`	$\leqslant 1R$	–
* May only be used in partial class definitions		

Table 4.3: Features of OWL Lite present in OWL Lite⁻

and an inconsistency in the program is derived. Notice that if a Datalog engine does not explicitly allow the use of integrity constraints, these can be easily emulated by introducing a special predicate symbol ic in the head of each integrity constraint. If the extension of the ic predicate is non-empty, this means the integrity constraint is violated.

- LP_2: Datalog(IC,=), which additionally allows equality (also in rule heads). Note that equality in Datalog significantly increases the complexity of the implementation and query answering can be expected to be harder when equality is allowed[1].

- LP_3: Prolog(IC,=), which generalizes LP_2 by allowing function symbols and unsafe rules. Note that we do not allow negation.

Notice that our layering of the languages LP_0, LP_1, LP_2 and LP_3 differs slightly from the layering of the languages $\mathcal{LP}_0$, $\mathcal{LP}_1$, $\mathcal{LP}_2$ and $\mathcal{LP}_3$ presented by Volz [Volz, 2004]. In his layering, Volz first introduced equality, before introducing integrity constraints. We have chosen to first introduce integrity constraints before introducing equality, because equality (in the head of the rule) comes at a much higher performance penalty that integrity constraints.

We will discuss all the features of OWL Lite and say whether they are in OWL Lite⁻ and discuss any limitations on their use, such as the restricted use of the top ($\top$; corresponds to `owl:Thing`) and bottom ($\bot$; corresponds to `owl:Nothing`) concepts. Furthermore, we discuss for each feature the implications that inclusion of the feature would have on the complexity of the language.

[1]Note that most commercial Datalog implementations only allow equality in the body of the rule, which disallows deriving equality, ruling out a lot of the computational problems.

We discuss the benefits and drawbacks of including each feature, that is currently not in OWL Lite⁻, besides the general drawback, that features which require a more expressive Logic Programming variant make reasoning more expensive.

The resultant complexity of the language required for a particular feature, as outlined in the following section, are based on [Volz, 2004, Chapter 4].

Partial class definitions Partial class definitions are allowed in OWL Lite⁻ with the restriction that `owl:Nothing` ($\bot$) is not allowed to occur on the right-hand side (the defining part) and `owl:Thing` ($\top$) is not allowed to occur on the left hand side (the defined part), i.e. it is not allowed to redefine `owl:Thing`.

Having `owl:Thing` on the left-hand side of the definition makes every class and restriction C_i equivalent to `owl:Thing`, which amounts to specifying rules, which are always true. Having `owl:Nothing` on the right-hand side of the definition makes class C equivalent to `owl:Nothing`, which amounts to having an integrity constraint, which basically states that C is not allowed to have any instances. Arbitrary integrity constraints can now be constructed by stating equivalence between C and the expression to be put in the integrity constraint. An example of the possible use of such an integrity constraint could be stating that a person with two legs is not allowed to have four legs, which can be written down as the Description Logic statement $\exists hasLegs.2 \sqcap \exists hasLegs.4 \sqsubseteq \bot$[2].

As was shown by Volz, `owl:Thing` is in LP_2. Because the use of `owl:Nothing` amounts to an integrity constraint, it is expressible in LP_1. The symbol $false$, resulting from the translation presented in Table 4.1, occurs in the head of the rule of the integrity constraint. It can be verified that if the extension of $false$ is non-empty, the constraint is violated.

Complete class definitions Complete class definitions are allowed in OWL Lite⁻ with the restriction that `owl:Nothing` ($\bot$) and `owl:Thing` ($\top$) are not allowed to occur in the definition.

Complete class definitions with `owl:Nothing` on the left-hand side are a way to express disjointness of classes in OWL Lite. A complete class definition with `owl:Nothing` on the left hand side and a number of classes $C_1 \dots C_n$ on the right-hand side is equivalent to the `DisjointClasses(`$C_1 \dots C_n$`)` statement in OWL DL.

Stating disjointness of classes is useful in cases such as the second example of Chapter 2. However, when treating a range restriction as a "real" constraint[3], this would not be necessary, because the constraint would be violated if `belle` could not be inferred to be a member of `Person`.

Use of `owl:Thing` on the left-hand side is equivalent to stating that each class on the right-hand side is equivalent to `owl:Thing`. Use of `owl:Thing` on the right-hand side would actually only cause problems when it is the only description there. If there are any other descriptions on the right-hand side, `owl:Thing` is removed during the process of normalization.

Similarly as with partial class definitions, including `owl:Thing` in the language would require LP_2 and including `owl:Nothing` would require LP_1.

[2]Two remarks are in order about this example. First, the above expression is not valid in $\mathcal{SHIF}$, the language underlying OWL Lite, since it uses nominals in the class definition. Second, it is not possible in OWL to restrict the number of legs to two in this example, because OWL does not allow predicates over data types. See chapter 5 for a more elaborate discussion of data types in OWL.

[3]By a "real" constraint we mean a constraint in the sense of a database constraint, which does not infer equality.

Class equivalence Class equivalence can be expressed in OWL Lite⁻, but the use of `owl:Thing` and `owl:Nothing` is disallowed.

Class equivalence can be reduced to complete class definitions with only one concept on the right-hand side, therefore the same argument holds with respect to the uses of `owl:Thing` and `owl:Nothing`. Note that class equivalence does not require equality in the language. Translating class equivalence simply requires two implications.

Property definitions OWL Lite property definitions along with the specification of subsuming properties are allowed in OWL Lite⁻.

Domain and Range restrictions Domain (range) restrictions are allowed as long as the domain (range) is not `owl:Nothing`. If the domain (or range) is restricted to `owl:Thing`, the restriction can be eliminated, since it would actually not be a restriction at all.

Inverse properties Inverse properties can be expressed in OWL Lite⁻.

Symmetric properties Symmetric properties can be expressed in OWL Lite⁻.

Functional and Inverse Functional properties (Inverse) functional properties can not be expressed in OWL Lite⁻. As can be seen in Table 4.2, functionality of properties requires equality in the language, which is not part of Datalog; therefore functional properties are not in OWL Lite⁻. Note that (Inverse) Functionality of properties is expressible in LP_2, because of the equality in the language.

(Inverse) Functionality of a property is equivalent to a maximal cardinality constraint of 1. As we have argued in chapter 2, the maximal cardinality restrictions in OWL Lite cause derivation of equality in a non-intuitive manner. From this perspective, the lack of functionality of properties in OWL Lite⁻ is not a big problem and arguably makes the language more intuitive and thus more usable.

Transitive properties Transitivity of properties can be expressed in OWL Lite⁻. In fact, in OWL Lite⁻, the specification of transitivity of properties is not restricted, as it is in OWL Lite, because there are no cardinality restrictions and there is no functionality of properties in OWL Lite⁻. The reason for the limitations on the specification of transitive properties in OWL Lite is that mixing transitive properties and cardinality restrictions introduces undecidability in Description Logics, as was shown in [Horrocks et al., 2000].

Subproperty descriptions Subsumption of properties can be expressed in OWL Lite⁻.

Property equivalence Equivalence of properties can be expressed in OWL Lite⁻.

Individual assertions Assertions of individuals can be expressed in OWL Lite⁻.

Property values Assertions of property values can be expressed in OWL Lite⁻.

Individual (in)equality Individual (in)equality cannot be expressed in OWL Lite⁻, because it needs equality in the language, which is not present in

Datalog. Inequality furthermore needs integrity constraints in the language. Therefore, individual (in)equality can be expressed LP_1.

On the one hand, strong equality support is required for working with the web. Simply take URIs as an example. Already the Unix file system offers an infinite number of ways to refer to the same file using different URIs. Therefore, in a web context we need to be able to deal with equality. On the other hand, the means offered by OWL to deal with this issue are completely inappropriate for this purpose (cf. [Fensel, 2003, pp. 45]). First, one can state a finite set of equalities which would take the modeler forever to simply capture the equalities in a Unix file system. Second, the modeler could capture equalities by cardinality constraints. For example, he could state a maximal cardinality of 3 for a role in OWL-DL (a maximality constraint of 1 in OWL-Lite) and if the system found for example 4 role fillers the system would infer that enough of these identifiers were equal to ensure that there were not more than 3 role fillers. The system would neither tell you which of them were equal nor would we recommend you to trust this inference. In a nutshell, the equality inference of OWL is (1) dangerous, (2) inappropriate, and (3) inflates the inference costs without any value in return. It is dangerous because simple modeling mistakes are used to infer even more erroneous facts. It is inappropriate because none of the required equality mechanisms for literals and URIs can be expressed in it. It is costly because it replaces simple syntactical term unification by complex semantic satisfiability reasoning.

For OWL Lite⁻ the presence or absence of the unique name assumption (UNA) has no implications, because there is no equality. However, when extending the language to allow for cardinality restrictions and functionality of properties, the choice of whether to adhere to the UNA has semantic implications. Remarkably, the Description Logic reasoner RACER, which is currently among the most popular DL reasoners, adheres to the UNA.

Primitive classes Primitive classes can be expressed in OWL Lite⁻.

owl:Thing `owl:Thing` is translated to the symbol *true* in the translation to First-Order Logic (see Table 4.1). As we have seen above, this symbol can often be eliminated after the translation. However, when the symbol cannot be eliminated, special support in the Datalog engine is required to evaluate rules which include the *true* symbol.

Therefore, in the general case `owl:Thing` is not allowed in OWL Lite⁻, although it is in special cases, where it can be eliminated. It is also not hard to extend any datalog implementation with support for the *true* symbol.

owl:Nothing `owl:Nothing` is translated to the symbol $false$ in the translation to First-Order Logic (see Table 4.1). As we have seen above, this symbol can often be eliminated after the translation. However, when the symbol cannot be eliminated, special support in the Datalog engine is required to evaluate rules which include the $false$ symbol.

Therefore, in the general case `owl:Thing` is not allowed in OWL Lite⁻, although it is in special cases, where it can be eliminated. It is also not hard to extend any Datalog implementation with support for the $false$ symbol. In general, use of the $false$ symbol amounts to integrity constraints, which are included in LP_1.

Existential value restrictions In Datalog, all variables are universally quantified; the existential quantifier is not part of Datalog. In a First-Order

formula, the existentially quantified variable can be eliminated through skolemization, i.e. by substituting it with a skolem function. However, function symbols are not allowed in Datalog.

Since LP_3 allows the unrestricted use of function symbols, the existential value restriction can be represented in this language, both in partial and complete class definitions. Introducing function symbols in logic programs leads to undecidability in the general case, although some restrictions can make the program decidable (cf. [Dantsin et al., 2001], [Bonatti, 2004]).

Note that in $\mathcal{L}_0$ [Volz, 2004], existential restrictions are allowed on the left-hand side of the inclusion symbol. OWL Lite, however, only allows named classes on the left-hand side (i.e. partial class definitions), therefore we do not have to consider this case here.

As was shown in [Volz, 2004], existential value restrictions are not widely used in currently available ontologies and we believe that this feature would not be intuitive to use for people with a database or a programming background.

Universal value restrictions OWL Lite⁻ allows universal value restrictions in partial class definitions, i.e. only on the right-hand side of the inclusion axiom. Furthermore, the use of `owl:Nothing` is not allowed in a universal restriction, because it would require the use of integrity constraints. Therefore, universal value restrictions with `owl:Nothing` in partial class definitions can be expressed in LP_1.

Universal value restrictions in complete class definitions cannot in general be represented in any of the considered logic programming formalisms, because universal restrictions on the left-hand side of the inclusion symbol are translated to a disjunction with more than one positive literal, which is no longer a Horn formula. Disjunctive Logic Programming, an extension of logic programming, which allows disjunction in the head of the rule, has been shown to be able to express universal restriction on the left-hand side of the inclusion symbol [Hustadt et al., 2004]. However, introducing disjunction in the head in Logic Programming significantly increases complexity. Answering positive boolean queries in disjunctive datalog is co-NP-complete in the data size [Dantsin et al., 2001]. We must note here that this additional complexity is only introduced when you actually use the disjunction in the head. When not using the disjunction in the head, data complexity is still in P. Thus, you only pay the performance penalty when actually using the additional expressiveness. However, if the additional expressiveness is there, people are more likely to use it, even if it is not required.

Minimal cardinality 0 A minimal cardinality restriction of 0 is equivalent to `owl:Thing`, which can be eliminated in most class definitions. However, for a complete class definition, when `owl:Thing` appears as the only class on the right-hand side, the axiom can no longer be translated to plain Datalog. Therefore, we disallow the use of the minimal cardinality restriction in complete class definitions.

It can be argued that the minimal cardinality restriction of 0 should be left out of the language, because it has no real use. When asserting that a property has a cardinality of at least 0, no restriction is actually imposed on the property.

Minimal cardinality 1 The minimal cardinality restriction of 1 can not be expressed in OWL Lite⁻, because of the need for existential quantification

and the associated skolemization and introduction of skolem functions, as for the existential value restrictions. Actually, a minimal cardinality restriction of 1 can be seen as a generalized form of an existential value restriction. It is actually equivalent to an existential value restriction with a range of `owl:Thing`. A minimal cardinality of 1 can also be expressed in LP_3, because of this equivalence, as with the existential value restriction, both in partial and complete class definitions.

Maximal cardinality 0 A maximal cardinality restriction of 0 is actually equivalent to a universal value restriction with `owl:Nothing` as its range. As we have seen above, this cannot be expressed in OWL Lite⁻. As we have also seen above, it would be possible to express a maximal cardinality of 0 in a partial class definition in LP_1.

The usefulness of a maximal cardinality restriction of 0 is disputable. Why define a property when you do not want any property fillers? The only use of this kind of restriction is that it could be used to say that individuals of a particular class *may not* have this particular property. We expect, however, that the use of this kind of construct will be limited.

Maximal cardinality 1 A maximal cardinality restriction of 1 is equivalent to stating that a property is functional. As we have already shown, this cannot be expressed in OWL Lite⁻. Maximal cardinality of 1 is expressible in LP_2, but only in partial class definitions, because maximal cardinality on the left-hand side of the inclusion symbol would lead to the introduction of function symbols and disjunction in the head (when using the function π), which cannot be handled in traditional logic programming languages.

As we have already argued in Chapter 2, the cardinality restrictions in OWL Lite lead to the non-intuitive deduction of equality. We envision that future extensions of OWL Lite⁻ will introduce database-style cardinality constraints.

Summarizing, the abstract syntax of OWL Lite⁻ is exactly the abstract syntax of OWL Lite as presented in [Patel-Schneider et al., 2004], leaving out all the constructs indicated with a '–' in Table 4.3 and restricting the use of the `allValuesFrom` property restriction.

4.3 OWL Lite⁻ RDF Syntax

Table 4.3 identifies exactly which parts of the **abstract syntax** of OWL Lite are in OWL Lite⁻. In order to obtain the **RDF graph** corresponding to an OWL Lite⁻, one can simply apply the transformations presented in [Patel-Schneider et al., 2004, section 4.1] to the abstract syntax.

The definition of OWL Lite⁻ Ontologies in RDF graph form is as in [Patel-Schneider et al., 2004, section 4.2] augmented with the following definition:

Definition 1. *An RDF graph is an **OWL Lite⁻ ontology in RDF graph form** if it is the result of the transformation to triples (cf. [Patel-Schneider et al., 2004, section 4.1]) of a collection of OWL Lite⁻ ontologies, axioms and facts in abstract syntax form that has a separated vocabulary*[4]*.*

[4]Informally, the vocabulary is separated if the sets of IDs for the classes, individuals, properties, property values, etc. are disjoint, if every individual has a type and if the RDF and OWL vocabularies are used in a restricted way. For a normative definition, see [Patel-Schneider et al., 2004, section 4.2]

OWL Lite⁻ Abstract Syntax	Datalog
`Class(`C `partial` D_1 ... D_n`)`	$C(y) \rightarrow \bigwedge \phi_{\mathcal{LP}}(D_i, y)$
`Class(`C `complete` D_1 ... D_n`)`	$\begin{cases} C(y) \rightarrow \bigwedge D_i(y) \\ \bigwedge D_i(y) \rightarrow C(y) \end{cases}$
`EquivalentClasses(`C_1 ... C_n`)`	$\begin{cases} C_i(y) \rightarrow C_j(y) \\ C_j(y) \rightarrow C_i(y) \end{cases}, i \neq j$
`ObjectProperty(`R `super(`R_1`)...super(`R_n`)`	$R(x,y) \rightarrow \bigwedge R_i(x,y)$
`domain(`C_1`)` ... `domain(`C_n`)`	$R(x,y) \rightarrow \bigwedge C_i(x)$
`range(`C_1`)` ... `range(`C_n`)`	$R(x,y) \rightarrow \bigwedge C_i(y)$
`[inverseOf(`R_0`)]`	$\begin{cases} R(x,y) \rightarrow R_0(y,x) \\ R_0(x,y) \rightarrow R(y,x) \end{cases}$
`[Symmetric]`	$R(x,y) \rightarrow R(y,x)$
`[Transitive])`	$R(x,y) \wedge R(y,z) \rightarrow R(x,z)$
`SubPropertyOf(`R_1 R_2`)`	$R_1(x,y) \rightarrow R_2(x,y)$
`EquivalentProperties(`R_1 ... R_n`)`	$\begin{cases} R_i(x,y) \rightarrow R_j(x,y) \\ R_j(x,y) \rightarrow R_i(x,y) \end{cases}, i \neq j$
`Individual(`o `type(`C_1`)` ... `type(`C_n`)` `value(`R_1 o_1`)` ... `value(`R_n o_n`))`	$\bigwedge C_i(o)$ $\bigwedge R_i(o, o_i)$
`restriction(`R `allValuesFrom(`C`))`	$R(x,y) \rightarrow C(y)$

Table 4.4: Translating OWL Lite⁻ Abstract syntax into Datalog

4.4 OWL Lite⁻ Model-Theoretic Semantics

Because OWL Lite⁻ is a strict subset of OWL Lite and thus OWL DL, the direct model-theoretic semantics of OWL Lite⁻ corresponds with the semantics for OWL DL in [Patel-Schneider et al., 2004, section 3] with the restriction of the abstract syntax to the OWL Lite⁻ abstract syntax.

4.5 Transforming OWL Lite⁻ to Datalog

We adopt the function ϕ, presented in [Volz, 2004, pp. 120, 122], for the translation of OWL Lite⁻ class constructors, properties and assertions to Datalog to show the correspondence between OWL Lite⁻ axioms and assertions in Table 4.4. Note that the function ϕ still occurs in the translation of the partial class definition. This is because it is possible to either put a named class or a universal property restriction there. A class C is translated to a unary predicate $C(x)$; a universal restriction is translated according to the last line of Table 4.4.

Each rule resulting from the translation presented in table 4.4 can be easily translated into pure Datalog by applying Lloyd-Topor transformations [Lloyd and Topor, 1984]. In fact, the only transformations that need to be done after the transformations presented in Table 4.4 is splitting the rules with conjunctions in the head into a set of rules, each with the same body, but in each head a different literal from the original rules.

4.6 The relation between OWL Lite⁻ and RDFS

RDF Schema (RDFS) [Brickley and Guha, 2004] is a light-weight ontology language, consisting of classes, class hierarchies, properties, property hierarchies and domain and range restrictions. RDFS was developed as a vocabulary description language for RDF. The combination of RDF and RDFS is usually seen as the lowest layer in the Semantic Web languages. More expressive languages such as OWL are layered on top of RDF(S). OWL Full is strictly layered on top

of RDF(S). However, both OWL Lite and OWL DL pose several restrictions on the use of RDFS in order to make the RDFS document valid OWL Lite/DL [Patel-Schneider et al., 2004]. In this section we will explain the relationship between OWL Lite⁻ and RDFS.

OWL Lite is (partly) syntactically and (partly) semantically layered on top of RDFS. There is, however, a subset of RDFS which is both syntactically and semantically included in OWL Lite. We will show that this RDFS subset of OWL Lite is the same as the RDFS subset of OWL Lite⁻.

We will first describe the features of RDFS, which are in OWL Lite⁻. Then, we will describe the restrictions OWL Lite⁻ imposes on RDFS. Finally, we will describe what expressivity OWL Lite⁻ adds on top of RDFS. It turns out that only in the last aspect (the expressivity added by OWL Lite⁻) does OWL Lite⁻ differ from OWL Lite, although from a practical viewpoint, the expressivity does not differ that much.

4.6.1 RDFS in OWL Lite⁻

It turns out that the RDFS subset of OWL Lite⁻ corresponds with the RDFS subset of OWL Lite. OWL Lite captures the following features of RDF Schema [McGuinness and van Harmelen, 2004]:

Classes and class hierarchies RDFS classes and the class hierarchy are captured by the partial class definitions (the first line in Table 4.3). The limitations on the use of `owl:Thing` and `owl:Nothing` have no implications for the RDFS subset of OWL Lite⁻, since neither is in RDFS.

Properties and property hierarchies Properties and property hierarchies are captured in OWL Lite⁻ by the property definitions and the `SubPropertyOf` statement, although the use of the latter is not necessary. The inclusion of `SubPropertyOf` in OWL Lite seems kind of strange, because the construct `SubClassOf` has been omitted from OWL Lite (it is in OWL DL) and property subsumption can already be expressed in the property definitions themselves.

We think that from a syntax point of view it does make sense to group superclasses of a particular class in the class definition itself, instead of writing separate `SubClassOf` axioms, because it groups the information about the class into one definition, so that the modeler does not have to look through the entire ontology to find information regarding the particular class of interest. We wonder why the same thing was not done for the `SubPropertyOf` construct[5].

Domain and range restrictions Domain and range restrictions in OWL Lite⁻ are equivalent to those in OWL Lite, except that OWL Lite⁻ does not allow `owl:Nothing` in the domain (range). This is again not a problem with respect to the RDFS subset, because `owl:Nothing` is not in RDFS.

Individuals As in OWL Lite, it is possible to assert individuals as members of classes and property fillers. Therefore, with respect to the RDFS subset, the treatment of individuals is the same in OWL Lite⁻ as it is in OWL Lite.

[5]We could have omitted the `SubPropertyOf` construct from the OWL Lite⁻ language, but we chose not to do so, because our goal was to have the maximal possible subset of OWL Lite, which can be translated to Datalog. In future work, we will re-evaluate the constructs in the language and we may decide to drop this construct from the syntax.

4.6.2 Restrictions on RDFS imposed by OWL Lite⁻

OWL Lite, and thus OWL Lite⁻, poses several restrictions on the use of RDFS, most notably:

- The OWL vocabulary cannot be used for identifiers for classes, properties or individuals and the sets of identifiers used for classes, properties and individuals must be disjoint, in other words the vocabulary must be *separated*, see also section 4.3.
- There are restrictions on the RDF graph in the case of OWL restrictions, because, intuitively, the modeling of an OWL restriction consumes several triples that must occur together in a particular form.

The first point requires the sets of classes and individuals to be disjoint, which means that the same resource cannot be seen both as a class and an individual in OWL Lite (and thus OWL Lite⁻). However, treating classes as instances has been identified as a useful feature for an ontology language for the Semantic Web [Schreiber, 2002]. We will revisit this point in our discussion of future work; we plan to allow the treatment of classes as instances in order to make the language truly useful and usable on the Semantic Web.

4.6.3 Expressivity of OWL Lite⁻ compared to RDFS

Since OWL Lite⁻ is a restricted subset of OWL Lite, just like RDFS, it is useful to see if the language really differs from RDFS and whether it really adds significant expressivity in order to justify the development and use of the language.

The features of OWL Lite⁻ that are not in RDFS are, in short:

- Complete (i.e. necessary and sufficient) class definitions, whereas RDFS only allows partial class definitions.
- Equivalence between classes and properties. In fact, this feature is semantically possible already in RDFS, but the syntax was missing. Stating `A subClassOf B` and `B subClassOf A` in RDFS is equivalent to `EquivalentClasses(A B)` in OWL Lite⁻.
- Inverse and symmetric properties.
- Transitive properties.
- Value restrictions in partial class definitions. In RDFS it is not possible to have local range restrictions for properties. In OWL Lite⁻ it is, through the universal value restriction.

The features that are furthermore in OWL Lite, which are not in OWL Lite⁻ are, besides the more general use of `owl:Thing` and `owl:Nothing`, (inverse) functional properties, (in)equality assertions of individuals, existential value restrictions, universal value restrictions in complete class definitions, minimal cardinality restrictions and maximal cardinality restrictions. But, as we have shown in section 4.2, most of these additional features are not intuitive in their usage and come at the cost of a significant increase in computational complexity.

4.7 Summary

In this chapter, we have introduced OWL Lite$^-$, a proper subset of OWL Lite, which can be translated into the deductive database language Datalog.

We have provided a justification for the omission of certain features from the language and have shown that there was a good reason for many of these features not to be included in the OWL Lite$^-$ language, because many of the omitted features introduce unwanted equality reasoning and nonintuitive deduction of equality.

For the discovery of the OWL Lite$^-$ fragment of OWL Lite, we have relied heavily on [Grosof et al., 2003], which was elaborated on by Raphael Volz in his PhD dissertation [Volz, 2004]. We have taken the intersection of the $\mathcal{SHOIN}$ description logic language, which underlies OWL DL, and Logic Programming, called Description Logic Programs, and most notable the language $\mathcal{L}_0$, which is the intersection of $\mathcal{SHOIN}$ and the popular deductive database language Datalog. We have identified that nearly all features in $\mathcal{L}_0$ are in OWL Lite, thus, we could use nearly all of the $\mathcal{L}_0$ language and were able to reuse the work by Raphael Volz for the most part.

We have restricted both the OWL Lite abstract syntax and the OWL Lite RDF syntax to OWL Lite$^-$ abstract and RDF syntax, respectively. We have also presented a direct translation from OWL Lite$^-$ to Datalog. Furthermore, we have shown the relationship between OWL Lite$^-$ and RDFS and have shown that OWL Lite$^-$ is a proper extension of a (OWL Lite) restricted form of RDFS. We have furthermore shown that OWL Lite$^-$ adds significant expressive power on top of RDFS, which justifies the existence of the language.

Because OWL Lite$^-$ can be translated directly to Datalog, the language could be used as the basis for many extensions that have been investigated in the area of Logic Programming. Furthermore, after translating an OWL Lite$^-$ ontology to Datalog, building rules on top of the ontology is relatively straightforward.

Note also that, because it is not possible to derive negative information from a plain Datalog program, it is also not possible to derive negative information from an OWL Lite$^-$ ontology. In other words, it is not possible to have an inconsistency in an OWL Lite$^-$ ontology. Furthermore, because we have taken the OWL Lite subset of $\mathcal{L}_0$, we do not allow the use of the `hasValue` property restriction ($\exists R.\{n\}$).

5 Conclusions and Future Work

We have presented the ontology language OWL Lite⁻, which is a variant of the OWL Lite species of the Web Ontology Language OWL. We have used the work of Raphael Volz [Volz, 2004] as a basis for identifying a useful and usable subset of OWL Lite, which allows for efficient reasoning over instances in a Semantic Web context.

After having identified the maximal subset of OWL Lite, which can be evaluated efficiently (in the context of query answering) using a Logic Programming engine, we can think of ways of extending this language with additional features that might be useful for knowledge representation.

Furthermore, OWL Lite⁻ can straightforwardly be extended with rules in order to allow for the description of behavior, for example of Semantic Web Services [Fensel and Bussler, 2002], which require the description of the relationships between inputs and outputs, which requires chaining variables over predicates.

5.1 Datatypes in OWL Lite⁻

In this section we first briefly describe the datatype support in OWL and then describe ways of handling datatypes in OWL Lite⁻.

5.1.1 Datatypes in OWL

OWL allows for a limited treatment of datatypes. In order to support datatypes, an OWL Lite ontology is interpreted in two disjoint domains: the abstract domain and the concrete domain. An OWL Lite reasoner deals only with the abstract domain and assumes a datatype oracle, which is assumed to have a sound and complete decision procedure for the emptiness of an expression of the form $d_1^{\mathbf{D}} \cup \ldots \cup d_n^{\mathbf{D}}$ where d_i is a (possibly negated) concrete data type from the domain $\mathbf{D}$ [Horrocks and Sattler, 2001].

The three major limitations of datatype support in OWL are [Pan and Horrocks, 2004]:

- OWL does not support negated datatypes.
- OWL does not support the use of datatype predicates. In OWL, it is only possible to refer to a single value in a datatype domain. It is, for example, not possible to express the greater-then ($\geq$) relation for the `xsd:integer` domain in OWL. It is therefore not possible to express, for example, then an adult is at least 18 years old, although this is possible in most investigated Description Logic extensions with concrete domains (e.g. [Horrocks and Sattler, 2001; Baader and Hanschke, 1991]), where such an axiom can be easily expressed: $Adult \sqsubseteq \forall.hasAge.\geq_{18}$. Here, $\geq_{18}$ denotes all integers greater than or equal to 18 (this is actually a user-defined data type).
- OWL does not support user-defined datatypes. One would expect that OWL does support user-defined datatypes, especially because it uses the simple datatypes from XML Schema. In XML Schema it is possible for

the user to define datatypes, however, these datatypes can not be used in OWL, because there is no proper way to reference them.

[Pan and Horrocks, 2004] shows an extension of OWL with so-called datatype groups. Datatype groups overcome the aforementioned limitations of datatype support in OWL and bridge the gap between datatypes in OWL and concrete domains as they have been investigated in the Description Logic community (see e.g. [Baader and Hanschke, 1991; Horrocks and Sattler, 2001; Lutz, 2002]).

A datatype group is essentially a group of disjoint datatypes, where each datatype forms a subgroup. A datatype group $\mathcal{G}$ consists of a predicate map M_p, which maps predicate URI references to predicates, a set of predicate URI references $D_\mathcal{G}$ and a domain function dom. A subgroup intuitively consists of all predicates defining relations on one particular datatype (e.g. *integer*). A subgroup is identified with a predicate URI, which is interpreted as a unary predicate with the entire datatype as its extension (e.g. `xsd:integer` corresponds to the datatype *integer*). Finally, [Pan and Horrocks, 2004] introduces a number of so-called $\mathcal{G}$-Datatype expressions, with which the user can construct new data types and new predicates, derived from the existing datatypes and predicates

5.1.2 Ways of handling datatypes in OWL Lite⁻

There currently exists a gap between the way data types are handled in OWL and the treatment of concrete domains in Description Logics. The latter only allows one concrete domain (e.g. *integer*), whereas the former allows many different data types (e.g. *string*, *date*, *integer*), albeit with many limitations. [Pan and Horrocks, 2004] shows a way to bridge the gap between the two, while extending datatype support in OWL, in order to allow the full expressiveness of concrete domains in Description Logics, while using different data types and retaining decidability. However, the datatype group approach still requires an external datatype oracle to evaluate the datatype expressions and, more specifically, to decide conjunctive queries for each datatype.

Many datalog implementations have support for concrete values built in, although there is typically no strong support for data typing. Most implementations, however, do provide a large number of built-in predicates[1]. One could view these built-in predicates as an oracle, since they are outside of the logical framework of the logic programming formalism. Many datalog engines therefore already provide a limited datatype oracle.

Extending the abstract syntax and semantics of OWL Lite⁻ with concrete domains is trivial, since OWL Lite⁻ is a strict subset of OWL Lite and the abstract syntax and semantics for concrete domains in OWL Lite have already been described. We can then assume, as in OWL, an external datatype oracle, which can determine satisfiability of a conjunction of data values, or even an oracles, as in [Pan and Horrocks, 2004], which can answer conjunctive datatype queries, allowing datatype predicates.

5.2 Possible Future extensions of OWL Lite⁻

We have identified a number of features as possible extensions of OWL Lite⁻:

[1] e.g. http://www.ontoprise.de/documents/tutorial_flogic.pdf, section 6

Making hidden features explicit Many features of the OWL DL species that are not explicitly present in OWL Lite can be expressed in OWL Lite via simple transformations. In fact, the only descriptions in OWL DL that cannot be expressed in OWL Lite are those containing individuals and cardinalities higher than one [Horrocks et al., 2003]. Some of these features, but by no means all, can also be expressed in OWL Lite⁻. For example, intersection can be expressed by creating a class definition from more than one class or property restriction. As can be seen in Table 4.3, this translates to an intersection statement in Description Logic syntax.

A possible (syntactic) extension of OWL Lite⁻ is to enrich the syntax to make these hidden features explicit in the language. A possible drawback of this could be that the language might become too complex for its users, which was also the original argument for leaving these features out of the syntax of OWL Lite.

Re-introducing the `hasValue` restriction The $\mathcal{SHIF}$ Description Logic underlying OWL Lite does not allow the use of individuals in concept descriptions. Therefore, the `hasValue` property restriction, which is in OWL DL, is not in OWL Lite. However, as was shown by [Volz, 2004], the `hasValue` restriction does not introduce additional complexity when performing query answering in deductive databases. Therefore, it might be useful to re-introduce this restriction. One could also think of using the `hasValue` restriction as the basis for the use of (overridable) default values of properties.

Unique Name Assumption First of all, we do not believe that the unique name assumption holds, as such, in the web. We already mentioned the Unix file system where different URIs can denote the same file. Therefore, strong means for resolving this aspect of the web are essential. However, we do not recommend to naively try to cover this by a badly thought out equality mechanism as provided by OWL. Neither finite equality statements nor non-intuitive cardinality constraints can provide this support at a logical and modeling level. These mechanisms add significant computational complexity to the logical language without adding anything in terms of these requirements. Assuming an external oracle that normalizes term structures then using unique name assumptions is a much more reliable way to go. On the one hand, full support for name resolution in a web context can be provided. On the other hand, the complexity of the logical language remains moderate. Finally, bizarre inferences based on wrongly interpreted cardinality constraints are prevented (cf. [Fensel, 2003, pp. 45]).

Treatment of Classes as Instances In RDF(S) [Lassila and Swick, 1999; Brickley and Guha, 2004] classes and instances are not disjoint. A resource can be both a class and an instance. This feature is still available in OWL Full, but not in the species OWL Lite and OWL DL. It seems quite likely that such a feature will be necessary for the Semantic Web [Schreiber, 2002] and there are currently several logical languages which support such a treatment of classes as instances (e.g. HiLog [Chen et al., 1993], F-Logic [Kifer et al., 1995], SKIF [Hayes and Menzel, 2001]).

Pan and Horrocks [Pan and Horrocks, 2003] have developed an alternate semantics for RDFS, called RDFS(FA), because the RDF semantics [Hayes, 2004] are often seen as very hard to understand and because they do not fit with the Description Logic-style semantics of OWL Lite and OWL DL. RDFS(FA) divides the interpretation of an ontology into layers, or *strata*. The names in the strata are disjoint, i.e. a name occurring in one stratum can not occur in a

different stratum. Furthermore, a term occurring in a particular stratum always refers to a set of terms in the stratum directly below it. The stratum 0 contains all individual names. Stratum 1 contains classes in the Description Logic sense, i.e. sets of instances. Stratum 2 contains classes of classes, which corresponds to constructs such as `rdfs:Class` and `rdf:Property`.

Intuitively, in RDFS(FA), each terms corresponds to an instance of a class in the stratum directly above it and to a class of instances in the stratum directly below it. Therefore, RDFS(FA) could possibly be used as the semantic basis for a classes-as-instances facility.

(Local) Closed World Reasoning In OWL, the Open World Assumption (OWA) holds. This means that from the absence of information, nothing is inferred. Under the Closed-World Assumption (CWA), from the absence of information, negative information is inferred. The OWA is used, for example, in classical (first-order) logic and Description Logic. The CWA is typically used in database applications and programming environments. It has been shown in the database community that a lot of useful conclusions can be drawn under the CWA, which could not have been drawn under the OWA. The lack of possibilities to apply the CWA will most likely restrict the usefulness of the ontology language for many applications. One way to have the benefit of closed-world reasoning while still having the open-world assumption as a starting point, is to do local closed-world reasoning [Etzioni et al., 1997], by explicitly declaring which part of the knowledge is complete. Note that this is similar to the use of the epistemic **K** operator, which indicates another way to bridge the gap between the logic programming and the description logic worlds.

Some of the above mentioned features all imply going beyond the Description Logic-style semantics of OWL and also beyond standard Datalog. Furthermore, *closed world reasoning* adds non-monotonic features to the language. Future work will have to show how these features can interact with the First-Order style semantics of Description Logics. The list of possible extensions mentioned above is certainly not exhaustive. Within the Logic Programming research community, many extensions of Logic Programming languages have been investigated, which we have not even begun to explore in the context of OWL Lite⁻.

Acknowledgements

We would especially like to thank the reviewer of the deliverable, Boris Motik, for useful comments and discussions around this deliverable

The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, and Esperonto; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program.

The editors would like to thank to all the members of the WSMO working group for their advice and input into this document.

Bibliography

[Ajtai and Gurevitch, 2001] Ajtai, M. and Gurevitch, Y. (2001). Datalog vs. first-order logic. *Journal of Computer and System Sciences*, 49(3):562–588.

[Baader et al., 2003] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

[Baader and Hanschke, 1991] Baader, F. and Hanschke, P. (1991). A scheme for integrating concrete domains into concept languages. Technical Report RR-91-10, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH.

[Bechhofer et al., 2002] Bechhofer, S., Horrocks, I., and Turi, D. (2002). Instance store - database support for reasoning over individuals. Available from http://instancestore.man.ac.uk/instancestore.pdf.

[Bonatti, 2004] Bonatti, P. A. (2004). Reasoning with infinite stable models. *Artificial Intelligence*, 156(1):75–111.

[Borgida, 1996] Borgida, A. (1996). On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 82(1–2):353–367.

[Brickley and Guha, 2004] Brickley, D. and Guha, R. V. (2004). RDF vocabulary description language 1.0: RDF schema. Recommendation 10 February 2004, W3C. Available from http://www.w3.org/TR/rdf-schema/.

[Calvanese et al., 1998] Calvanese, D., Giancomo, G. D., and Lenzerini, M. (1998). On the decidability of query containment under constraints. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS'98)*, pages 149–158.

[Chen et al., 1993] Chen, W., Kifer, M., and Warren, D. S. (1993). HILOG: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230.

[Dantsin et al., 2001] Dantsin, E., Eiter, T., Gottlob, G., and Voronkov, A. (2001). Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425.

[Dean and Schreiber, 2004] Dean, M. and Schreiber, G., editors (2004). *OWL Web Ontology Language Reference*. W3C Recommendation 10 February 2004.

[Donini et al., 1998a] Donini, F. M., Lenzerini, M., Nardi, D., Nutt, W., and Schaerf, A. (1998a). An epistemic operator for description logics. *Artificial Intelligence*, 100(1–2):225–274.

[Donini et al., 1998b] Donini, F. M., Lenzerini, M., Nardi, D., and Schaerf, A. (1998b). AL-log: integrating datalog and description logics. *Journal of Intelligent Information Systems*, 10:227–252.

[Donini et al., 2002] Donini, F. M., Nardi, D., and Rosati, R. (2002). Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic*, 3(2):177–225.

[Etzioni et al., 1997] Etzioni, O., Golden, K., and Weld, D. (1997). Sound and efficient closed-world reasoning for planning artificial intelligence. *Artificial Intelligence*, 89(1-2):113–148.

[Fensel, 2003] Fensel, D. (2003). *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce, 2nd edition.* Springer-Verlag, Berlin.

[Fensel and Bussler, 2002] Fensel, D. and Bussler, C. (2002). The web service modeling framework wsmf. *Electronic Commerce Research and Applications*, 1(2):113–137.

[Grosof et al., 2003] Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In *Proc. Intl. Conf. on the World Wide Web (WWW-2003)*, Budapest, Hungary.

[Haarslev and Möller, 2003] Haarslev, V. and Möller, R. (2003). Incremental query answering for implementing document retrieval services. In *Proceedings of the International Workshop on Description Logics (DL-2003)*, pages 85–94, Rome, Italy.

[Haarslev and Möller, 2004] Haarslev, V. and Möller, R. (2004). Optimization techniques for retrieving resources described in OWL/RDF documents: First results. In *Ninth International Conference on the Principles of Knowledge Representation and Reasoning (KR2004)*, pages 163–173, Whistler, BC, Canada.

[Hayes, 2004] Hayes, P. (2004). RDF semantics. Technical report, W3C. W3C Recommendation 10 February 2004. Available from http://www.w3.org/TR/rdf-mt/.

[Hayes and Menzel, 2001] Hayes, P. and Menzel, C. (2001). A semantics for the knowledge interchange format. In *IJCAI 2001 Workshop on the IEEE Standard Upper Ontology.*

[Horrocks et al., 2004] Horrocks, I., Li, L., Turi, D., and Bechhofer, S. (2004). The instance store: DL reasoning with large numbers of individuals. In *Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 31–40.

[Horrocks and Patel-Schneider, 2003] Horrocks, I. and Patel-Schneider, P. F. (2003). Reducing OWL entailment to description logic satisfiability. In *Proc. of the 2003 International Semantic Web Conference (ISWC 2003)*, Sanibel Island, Florida.

[Horrocks and Patel-Schneider, 2004] Horrocks, I. and Patel-Schneider, P. F. (2004). A proposal for an OWL rules language. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 723–731. ACM.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2003). From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics.* To appear.

[Horrocks and Sattler, 2001] Horrocks, I. and Sattler, U. (2001). Ontology reasoning in the SHOQ(D) description logic. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI2001).*

[Horrocks et al., 2000] Horrocks, I., Sattler, U., and Tobies, S. (2000). Practical reasoning for very expressive description logics. *Logic Journal of the IGPL*, 8(3):239–264.

[Hustadt et al., 2004] Hustadt, U., Motik, B., and Sattler, U. (2004). Reasoning for description logics around SHIQ in a resolution framework. Technical Report 3-8-04/04, FZI.

[Kifer et al., 1995] Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *JACM*, 42(4):741–843.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource description framework (RDF) model and syntax specification. W3c recommendation, W3C. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.

[Levy and Rousset, 1998] Levy, A. Y. and Rousset, M.-C. (1998). Combining horn rules and description logics in CARIN. *Artificial Intelligence*, 104:165 – 209.

[Lloyd, 1987] Lloyd, J. W. (1987). *Foundations of Logic Programming (2nd edition)*. Springer-Verlag.

[Lloyd and Topor, 1984] Lloyd, J. W. and Topor, R. W. (1984). Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240.

[Lutz, 2002] Lutz, C. (2002). *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, Aachen.

[McGuinness and van Harmelen, 2004] McGuinness, D. L. and van Harmelen, F. (2004). Owl web ontology language overview. Recommendation 10 February 2004, W3C. Available from http://www.w3.org/TR/owl-features/.

[Motik et al., 2004] Motik, B., Sattler, U., and Studer, R. (2004). Adding DL-safe rules to OWL DL. to be submitted.

[Motik et al., 2003] Motik, B., Volz, R., and Maedche, A. (2003). Optimizing query answering in description logics using disjunctive deductive databases. In *Proc. KRDB-2003, volume 79 of CEUR Workshop Proceedings (CEUR-WS.org/Vol79/)*.

[Pan and Horrocks, 2003] Pan, J. Z. and Horrocks, I. (2003). RDFS(FA): A DL-ised sub-language of RDFS. In *2003 International Workshop on Description Logics (DL2003)*.

[Pan and Horrocks, 2004] Pan, J. Z. and Horrocks, I. (2004). OWL-E: Extending OWL with expressive datatype expressions. IMG Technical Report IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester. Available from http://dl-web.man.ac.uk/Doc/IMGTR-OWL-E.pdf.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). OWL web ontology language semantics and abstract syntax. Recommendation 10 February 2004, W3C.

[Schreiber, 2002] Schreiber, G. (2002). The web is not well-formed. *IEEE Intelligent Systems*, 17(2). Contribution to the section Trends and Controversies: Ontologies KISSES in Standardization.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of Database and Knowledge-Base Systems, Volume I*. Computer Science Press.

[Volz, 2004] Volz, R. (2004). *Web Ontology Reasoning with Logic Databases*. PhD thesis, AIFB, Karlsruhe.