WSML Deliverable

D20.1 v0.2
OWL$^-$

WSML Working Draft – May 15, 2005

Authors:
Jos de Bruijn
Axel Polleres
Rubén Lara
Dieter Fensel
Editors:
Jos de Bruijn
Axel Polleres
Reviewers:
Boris Motik
Michael Kifer

Final version:
http://www.wsmo.org/TR/d20/d20.1/v0.2/20050515/
Latest version:
http://www.wsmo.org/TR/d20/d20.1/v0.2/
Previous version:
http://www.wsmo.org/2004/d20/d20.1/v0.2/20041210/

Abstract

This deliverable presents restricted variants of the OWL Lite, DL and Full species of the OWL ontology language, called OWL Lite$^-$, OWL DL$^-$ and OWL Full$^-$, respectively. OWL Lite$^-$ and OWL DL$^-$ are strict subsets of OWL Lite and OWL DL. They are limited in such a way that there exists a translation from both languages directly into dunction-free logic programs, i.e., Datalog. Thus, any OWL Lite$^-$ or OWL DL$^-$ ontology can be translated into Datalog. An ontology language for which a translation to Datalog exists has several advantages. Most notably, it can benefit from long-term research in the field of deductive databases resulting in highly optimized query answering engines, and allows for easy implementation of a rule and a query language on top of the ontology.

It turns out that most current ontologies fall inside the fragments defined by OWL Lite$^-$ or OWL DL$^-$. 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.

It turns out that there are features in OWL DL, which can still be included in the language, while still allowing for efficient query answering. Therefore, we present extensions of OWL Lite$^-$ in order to create OWL DL$^-$, together with its abstract syntax and model-theoretic semantics. Then, the in our view most useful feature of OWL Full, the possibility to treat classes as instances, can also be translated to a positive Datalog program. We describe a (Datalog subset of) F-Logic based extension of OWL DL$^-$ to create OWL Full$^-$.

The overall goal of OWL$^-$ is not necessarily to serve as an independent ontology language but rather to define a maximal subset of the original OWL variants which does not fall outside a logic programming framework in order to allow for a straight-forward combination with rule languages. We believe that the OWL$^-$ subset of OWL can serve as a starting point for a more usable Web Ontology language and extensions in various directions which tackle some of the limitations of OWL outlined in this deliverable: Extensions of OWL Full$^-$ include the addition of Datatypes and Database-like constraints which are discussed in more detail in Deliverable 20.3 of WSMO [de Bruijn et al., 2004].

Contents

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 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, 2003a]. The $\mathcal{SHIF}(\mathbf{D})$ language allows complex class descriptions, including conjunction, disjunction, negation, existential and universal value restrictions for roles, role hierarchies, transitive roles, inverse roles, a restricted form of cardinality restrictions (cardinality 0 or 1) and (limited) 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 an extension of OWL Lite, adding support for individual names in class descriptions (also called *nominals*), and allowing arbitrary cardinality restrictions[1]. Even though there exists an RDF-based syntax for OWL Lite and OWL DL, these languages are not proper extensions of RDF(S). In order to provide a language which is truly layered on top of RDF(S), OWL Full was created. 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. One of the reasons for this is that it allows arbitrary roles in number restrictions, which makes the logic undecidable [Horrocks et al., 2000]; it is conjectured, 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 and satisfiability of a $\mathcal{SHOIN}$ knowledge base is NExpTime-time complete. Furthermore, ABox reasoning (more specifically, checking class membership and instance retrieval) in most Description Logic reasoners is very expensive, because for each individual a separate Tableaux satisfiability check 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 exist several proposal towards 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) optimized 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.

[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 and [Hustadt et al., 2004, Chapter 9]

[3]For now we disregard concrete domains in our treatment of OWL Lite. We describe a datatype extension in [de Bruijn et al., 2004].

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 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 deliverable, 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 and, subsequently, the OWL DL$^-$ 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. OWL DL adds support for nominals, which is notoriously hard to deal with in a Description Logic context. In fact, we are not aware of any implementation of a complete decision procedure for $\mathcal{SHOIN}$.

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 implemen-

[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].

tations of (supersets 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, 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 extend OWL Lite$^-$ with the `value` restriction and several syntactical extensions, coming from OWL DL, to create OWL DL$^-$, which is a strict subset of OWL DL. OWL DL$^-$ is still in the intersection of $\mathcal{SHOIN}$ and Datalog as identified by [Volz, 2004].

Although OWL Full is often ignored in the Semantic Web literature when considering reasoning facilities due to its undecidability, we do see several useful features in the OWL Full species. We believe that one of the most useful features of OWL Full is the possibility it offers for an element to be both a class and an instance. In order to incorporate this feature, we provide an axiomatization of OWL DL$^-$ in F-Logic and use this as a basis for OWL Full$^-$, which lifts the restriction of the separation between classes and instances. It has been argued that lifting this restriction is necessary for many applications [Schreiber, 2002].

This deliverable is structured as follows. We briefly review Description Logics, Logic Programming, Description Logic Programs and F-Logic in Chapter 2. In Chapter 3 we identify the major limitations of OWL. In Chapter 4 we describe the actual OWL Lite$^-$ language. In Chapter 5 we introduce OWL DL$^-$. Chapter 6 describes OWL Full$^-$. We finish with conclusions and suggestions for future work in Chapter 7.

[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/

2 Preliminaries

In order to make this paper self-contained, we will first describe several preliminaries, necessary to explain and provide a rationale for 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].

2.1 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.

Concept axioms in the TBox are of the form $C \sqsubseteq D$ (meaning the extension of C is a subset of the extension of D; D is more general than C) or $C \equiv D$ (where $C \equiv D$ is interpreted as $C \sqsubseteq D$ and $D \sqsubseteq C$) with C and D (possibly complex) descriptions. Descriptions can be built from named concepts (e.g. A) and role restrictions (e.g. $\forall R.C$ denotes a universal value restriction), connected with negation ($\neg$), union ($\sqcup$) and intersection ($\sqcap$).

The Description Logic underlying OWL DL ($\mathcal{SHOIN}$) allows only simple role axioms of the form $R \sqsubseteq S$ with R and S named roles. Some description logics allow for more complex role axioms, e.g. $\mathcal{DLR}$ [Calvanese et al., 1998].

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.

[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

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 instances 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.

2.2 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:

$$p_1 \vee \neg n_1 \vee \ldots \vee \neg n_k \tag{2.1}$$

This formula can be rewritten in the following form:

$$p_1 \leftarrow n_1 \wedge \ldots \wedge n_k \tag{2.2}$$

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.* A *logic program* consists of a set of horn clauses.

There are two basic styles for defining the semantics of logic programs. The first is by providing a 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 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*. Notice that plain Datalog does not allow negation, although there are many well-known extensions of both default (e.g. [Przymusinski, 1986; Gelfond and Lifschitz, 1988; Gelder et al., 1988]) and classical negation (e.g. [Gelfond and Lifschitz, 1991]).

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 ExpTime.

Many well-known extensions for Datalog exist, such as different forms of negation (e.g. stratified or well-founded negation, negation under the stable semantics, etc.) 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]). Notice that plain Datalog does not support the equality (=) predicate, although it is possible to axiomatize a weaker form of equality (the *congruence* relation) in the program. However, as pointed out earlier, when allowing equality to occur in the head of a rule, simple term unification is no longer possible and more complex reasoning is required.

2.3 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 the Description Logic language $\mathcal{SHOIN}$ 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 of this distinction between the left- and the right-hand side is disjunction, which can be used 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 a basic ontology language on the Semantic Web:

- $\mathcal{L}_0$ contains the most frequently occurring Description Logics 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 reasoning tasks can be reduced to query answering in (non-recursive) 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 (non-recursive) Datalog and a Description Logics, 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.

2.3.1 The Description Logics Family $\mathcal{DHL}$

Based on the work presented in [Grosof et al., 2003] and [Volz, 2004] we define the new (non-standard) family of Description Logic languages $\mathcal{DHL}$.

The acronym DHL stands for “Description Horn Logic” and was first introduced in [Grosof et al., 2003] to denote a subset of the $\mathcal{SHOIN}$ Description Logic which is based on a (function-free) Horn subset (see Section 2.2) of First-Order Logic (FOL). The term DLP was also introduced in [Grosof et al., 2003] to denote Logic Programs based on DHL ontologies. In fact, the difference between DHL and DLP is that DLP as a Logic Program only allows for the entailment of ground facts and not the entailment of formulae. Horn Logic, as a subset of First-Order Logic, does allow the entailment of formulae. However, with respect to the entailment of ground facts, DHL and DLP are equivalent. [Volz, 2004] further develops DHL and DLP, but just uses the acronym DLP to indicate both. We have chosen to use the name $\mathcal{DHL}$ for the new family of Description Logics, because it stays closer to First-Order semantics. For each of the members of the $\mathcal{DHL}$ family, there is a corresponding DLP variant, which is equivalent to the $\mathcal{DHL}$ member with respect to entailment of ground facts, but does not allow entailment of non-ground formulae.

The rationale for distinguishing between $\mathcal{DHL}$ and DLP is that a Description Logic reasoner can do TBox reasoning with a $\mathcal{DHL}$ ontology and a Logic Programming implementation can do ABox reasoning with the DLP Logic Program obtained from the $\mathcal{DHL}$ ontology.

Volz in his thesis [Volz, 2004] has identified four different languages, starting with the language $\mathcal{L}_0$, which has a translation to pure Datalog (see Section 2.2), and extending it in several steps to $\mathcal{L}_3$, which has a translation to Prolog (i.e. Datalog with unsafe rules and function symbols) with equality and integrity constraints.

Like Volz, we define our family of languages $\mathcal{DHL}$ as a subset of the $\mathcal{SHOIN}$ and (function-free) Horn Logic. However, we take a slightly different approach and define the following layers of expressiveness, together with the $\mathcal{DHL}$ variant:

1. $\mathcal{DHL}$ is a subset of function-free Horn Logic.
2. $\mathcal{DHL}_{\top,\bot}$ is a subset of function-free Horn Logic with the use of top ($\top$) and bottom ($\bot$)
3. $\mathcal{DHL}_{\top,\bot,\exists}$ is a subset of Horn Logic with the use of top ($\top$) and bottom ($\bot$), allowing function symbols

Because each variant of $\mathcal{DHL}$ is a subset of $\mathcal{SHOIN}$, they remain decidable with respect to the satisfiability problem. The third $\mathcal{DHL}$ variant, called $\mathcal{DHL}_{\top,\bot,\exists}$ thus also remains decidable, although it is a subset of full Horn Logic, which allows function symbols. In fact, these function symbols are used to "simulate" the existential quantifier in $\mathcal{SHOIN}$, which explains the $\exists$ symbol in the name.

$\mathcal{DHL}$

Pure $\mathcal{DHL}$ corresponds with Volz' $\mathcal{L}_0$ language with the extension of enumeration on the left-hand side of the general class inclusion symbol (GCI) "$\sqsubseteq$", see Corollary 2.1, Section 5.1).

Because we distinguish between descriptions that are allowed on the left-hand side and descriptions that are allowed on the right-hand side, we identify three types of descriptions, namely descriptions that are allowed on both sides of the GCI, simply referred to as $\mathcal{DHL}$ descriptions. Descriptions only allowed on the left- or the right hand side are referred to as $\mathcal{DHL}^L$ and $\mathcal{DHL}^R$, respectively.

Definition 2.1. *$\mathcal{DHL}$ descriptions are of the following form:*

A	*(named class)*
$C \sqcap D$	*(intersection)*
$\exists R.o$	*(hasValue restriction)*

Definition 2.2. *$\mathcal{DHL}^L$ descriptions are either $\mathcal{DHL}$ descriptions or are descriptions of the following form:*

$C \sqcup D$	*(union)*
$\exists R.C$	*(existential restriction)*
$\geqslant 1R$	*(minimal cardinality of 1)*
$\{o_1, \ldots, o_n\}$	*(enumeration)*

Definition 2.3. *$\mathcal{DHL}^R$ descriptions are either $\mathcal{DHL}$ descriptions or are descriptions of the following form:*

$\forall R.C$	*(universal restriction)*

We define the notion of a $\mathcal{DHL}$ TBox as follows:

Definition 2.4. *A $\mathcal{DHL}$ TBox is a set of TBox axioms, where C is a $\mathcal{DHL}^L$ description, D is a $\mathcal{DHL}^R$ description, E and F are $\mathcal{DHL}$ descriptions and R, S are properties. The following are valid $\mathcal{DHL}$ TBox axioms*[2]*:*

$C \sqsubseteq D$	*(class subsumption)*
$E \equiv F$	*(class equivalence)*
$R \sqsubseteq S$	*(property subsumption)*
$R \equiv S$	*(property equivalence)*
$R \equiv S^-$	*(property inverse)*
$R^+ \sqsubseteq R$	*(property transitivity)*
$\top \sqsubseteq \forall R^-.D$	*(property domain)*
$\top \sqsubseteq \forall R.D$	*(property range)*

[2]Note that we could safely additionally allow role intersection axioms of the form $R \sqcap S \sqsubseteq Q$) without leaving the Horn fragment of Description Logics, but we refrain from that since we aim at staying within a subset of $\mathcal{SHOIN}$

Besides a TBox, a $\mathcal{DHL}$ Knowledge Base can also contain an ABox:

Definition 2.5. *A $\mathcal{DHL}$ ABox is a set of ABox assertions, where C is a $\mathcal{DHL}^R$ description, R is a property and a and b are individuals. The following are valid $\mathcal{DHL}$ ABox assertions:*

$a \in C$	*(individual assertion)*
$\langle a, b\rangle \in R$	*(property filler)*

$\mathcal{DHL}_{\top,\bot}$

$\mathcal{DHL}_{\top,\bot}$ is a fairly simple extension of $\mathcal{DHL}$, allowing the $\top$ and $\bot$ concepts to occur in descriptions on both sides of the GCI. Thus, we obtain an updated definition for $\mathcal{DHL}_{\top,\bot}$ descriptions:

Definition 2.6. *$\mathcal{DHL}_{\top,\bot}$ descriptions are either $\mathcal{DHL}$ descriptions or are descriptions of the following form:*

$\top$	*(top)*
$\bot$	*(bottom)*

$\mathcal{DHL}^L_{\top,\bot}$ and $\mathcal{DHL}^R_{\top,\bot}$ can be defined similarly to Definitions 2.2 and 2.3.

$\mathcal{DHL}_{\top,\bot}$ is only a small extension of $\mathcal{DHL}$. However, the bottom concept can be used to express disjointness of descriptions and constraints in general, making it possible to derive negative information from the ontology, which is not possible in pure $\mathcal{DHL}$. TBox and ABox definitions are similar to $\mathcal{DHL}$.

$\mathcal{DHL}_{\top,\bot,\exists}$

$\mathcal{DHL}_{\top,\bot,\exists}$ is an extension of $\mathcal{DHL}_{\top,\bot}$ which allows the use of existential quantifiers on the right-hand side, thereby also a minimal cardinality of 1 on the right-hand side.

Definition 2.7. *$\mathcal{DHL}_{\top,\bot,\exists}$ descriptions are either $\mathcal{DHL}_{\top,\bot}$ descriptions or are descriptions of the following form:*

$\exists R.C \mid$ *(existential restriction)*

Definition 2.8. *$\mathcal{DHL}^R_{\top,\bot,\exists}$ descriptions are either $\mathcal{DHL}^R_{\top,\bot}$ descriptions or are descriptions of the following form:*

$\geqslant 1R \mid$ *(minimal cardinality 1)*

TBox and ABox definitions are similar to $\mathcal{DHL}_{\top,\bot}$.

$\mathcal{DHL}$ Semantics

In this section we define the semantics of $\mathcal{DHL}$ through a mapping to Horn logic. The translation is based on the translation of Description Logic to First-Order Logic in [Borgida, 1996]. The mapping is given in Table 2.1. The translations function π is applied recursively. x_{new} in the table denotes a new, previously unused variable in every translation step.[3] In our translation, all variables are implicitly universally quantified. Notice that not all resulting clauses are Horn. However, the formulas can be rewritten to pure Horn as follows by some post-processing:

- Double implication ($\leftrightarrow$) can simply be rewritten into two separate implications ($\leftarrow$ and $\rightarrow$).

[3] As opposed to [Borgida, 1996], we need more than two variables, since nested quantifiers are not allowed in Horn-Logic.

- Conjunction in the consequent of an implication can be replaced by multiple implications, i.e. $Body \rightarrow (H_1 \wedge \ldots \wedge H_n)$ becomes $(Body \rightarrow H_1) \wedge \ldots \wedge (Body \rightarrow H_n)$
- Disjunctions in the antecedent of an implication (from $\mathcal{DHL}^L$ expressions of the form $C_1 \sqcap \ldots \sqcap C_n$ or $\{o_1, \ldots, o_n\}$ can be eliminated similarly, cf. [Lloyd and Topor, 1984].
- Implications in the consequent of other implications (from $\mathcal{DHL}^R$ descriptions of the form $\forall R.C$ can be replaced recursively by replacing $Body \rightarrow (R(X,Y) \rightarrow C(Y))$ with $Body \wedge R(X,Y)C(Y)$ where X, Y are meta-variables.
- The use of the equality symbol $=$ (which also involves some post-processing) is further explained in Definition 2.9 below.

After this post-processing, we obtain a set of Horn-clauses, particularly function-free Horn-rules, i.e. theories in $\mathcal{DHL}$ even remain in the Datalog fragment of Horn Logic.

$\mathcal{DHL}$	**Horn logic**
Translating axioms	
$C \sqsubseteq D$	$\pi(C,x) \rightarrow \pi(D,x)$
$C \equiv D$	$\pi(C,x) \leftrightarrow \pi(D,x)$
$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
$R \equiv S$	$R(x,y) \leftrightarrow S(x,y)$
$R \equiv S^-$	$R(x,y) \leftrightarrow S(y,x)$
$R^+ \sqsubseteq R$	$R(x,y) \wedge R(y,z) \rightarrow R(x,z)$
$\top \sqsubseteq \forall R^-.D$	$R(x,y) \rightarrow \pi(D,x)$
$\top \sqsubseteq \forall R.D$	$R(x,y) \rightarrow \pi(D,y)$
Translating $\mathcal{DHL}$ descriptions	
$\pi(A,X)$	$A(X)$
$\pi(C_1 \sqcap \ldots \sqcap C_n, X)$	$\bigwedge \pi(C_i, X)$
$\pi(\exists R.o, X)$	$R(X,o)$
Translating $\mathcal{DHL}^L$ descriptions	
$\pi(C_1 \sqcup \ldots \sqcup C_n, X)$	$\bigvee \pi(C_i, X)$
$\pi(\exists R.C, X)$	$R(X, x_{new}) \wedge \pi(C, x_{new})$
$\pi(\geqslant 1R, X)$	$R(X, x_{new})$
$\pi(\{o_1, \ldots, o_n\})$	$\bigvee X = o_i$
Translating $\mathcal{DHL}^R$ descriptions	
$\pi(\forall R.C, X)$	$(R(X, x_{new}) \rightarrow \pi(C, x_{new}))$
Translating assertions	
$o \in A$	$A(o)$
$\langle o_1, o_2 \rangle \in R$	$R(o_1, o_2)$

X is a meta-variable and is substituted by the actual variable during the translation

Table 2.1: Mapping of $\mathcal{DHL}$ to Horn logic

Note that, we extend the translation of Volz [Volz, 2004] by enumeration $(\{o_1, \ldots, o_n\})$ on the left-hand side which *can* be expressed in Datalog without equality as well, where the translation rule from Table 2.1.

$$\pi(\{o_1, \ldots, o_n\}, X) \mapsto \bigvee X = o_i \qquad (2.3)$$

is an abbreviation. We define:

Definition 2.9. *In a first-order language without equality we define a formula of the form:*

$$A \leftarrow W_1 \wedge \ldots \wedge W_i \wedge (x = o_1 \vee \ldots \vee x = o_n) \wedge W_{i+1} \wedge \ldots \wedge W_n \tag{2.4}$$

where A, W_j are all positive literals, x is a (universally quantified) variable and o_k are all constants as abbreviation for the following set (i.e. conjunction) of formulas:

$$\begin{array}{r} (A \leftarrow W_1 \wedge \ldots \wedge W_i \wedge W_{i+1} \wedge \ldots \wedge W_n)[x := o_1] \ \wedge \\ \vdots \\ \wedge \ (A \leftarrow W_1 \wedge \ldots \wedge W_{i-1} \wedge W_{i+1} \wedge \ldots \wedge W_n)[x := o_n] \end{array}$$

where $F[x := o_i]$ is the formula obtained from a formula F by substituting all occurrences of variable x with the constant o_i.

We can now formulate the following conclusion (which contradicts the results of [Volz, 2004]):

Corollary 2.1. *Enumeration $\{o_1, \ldots, o_n\}$ on the left-hand side of the GCI can be represented in Horn logic.*

Proof. Following the transformation shows that the corresponding formula to $\{o_1, \ldots, o_n\} \sqsubseteq C$ is a conjunction of Horn formulae without $=$ in the language:

$$\pi(\{o_1, \ldots, o_n\} \sqsubseteq C, x)$$

becomes

$$\forall x.(\bigvee x = o_i \rightarrow C(x))$$

which, by Definition 2.9 yields

$$\bigwedge C(o_i)$$

□

As for $\mathcal{DHL}_{\top,\bot,\exists}$we have to extend Table 2.1 by the following lines (see Table 2.2) and a logically equivalent translation to Horn Logic is no longer possible in general.

$\mathcal{DHL}$	**First-Order Logic**
	Translating $\mathcal{DHL}_{\top,\bot,\exists}$ descriptions
$\pi(\top, X)$	$\top$
$\pi(\bot, X)$	$\bot$
	Translating $\mathcal{DHL}^R_{\top,\bot}$ descriptions
$\pi(\exists R.C, X)$	$(\exists x_{new} R(X, x_{new}) \wedge \pi(C, x_{new}))$

X is a meta-variable and is substituted by the actual variable during the translation

Table 2.2: Mapping of $\mathcal{DHL}_{\top,\bot,\exists}$ to First-Order logic

Also, for these additional translations, some post-processing is required, after the post-processing described above has been applied:

- Occurrences of $\top$ in the antecedent of an implication can be eliminated.
- Clauses with $\top$ in the consequent can be eliminated.

- Clauses with $\bot$ in the antecedent can be eliminated.
- Clauses with $\top$ in the consequent amount to constraints.
- By skolemizing all existentially quantified variables we can obtain an equisatisfiable (but not logically equivalent) set of Horn clauses.

2.4 Frame Logic

Frame Logic, or F-Logic [Kifer et al., 1995] provides a second-order, object-oriented-style syntax for a first-order logical language.

F-Logic was inspired by work on deductive databases, object oriented databases and object oriented programming.

An important concept in F-Logic is *object identity* [Khoshafian and Copeland, 1986]. Each object (i.e. class, instance, attribute) has a unique object identifier.

The basic syntax we are using here is taken from [Kifer et al., 1995]. To simplify matters, we focus only on the subset of F-Logic which we need for our translations. That is, we do not consider parameterized methods, schema molecules (used for signature definitions), functional (single-valued) methods and only use noninheritable attributes.

Formally, an F-Logic theory is a set of formulae constructed from so called *molecules*. Let σ denote the universe of possible object identifiers and $\mathcal{V}$ denote a (countably infinite) set of variables.

Definition 2.10 (Molecule). *A molecule in F-Logic is one of the following statements:*

1. *An* is-a *assertion of the form $C : D$ where $C, D \in \sigma \cup \mathcal{V}$*
2. *A* subclass-of *assertion of the form $C :: D$ where $C, D \in \sigma \cup \mathcal{V}$*
3. *Data molecules of the form $C[D \twoheadrightarrow E]$ where $C, D, E \in \sigma \cup \mathcal{V}$*

An F-Logic molecule is called ground, if it contains no variables.

Here, $C : D$ states that C is a member of class D whereas $C :: D$ states that C is a subclass of D. Data molecules of the form $C[D \rightarrow E]$ (or $C[D \twoheadrightarrow E]$, resp.) have the meaning that for individual C the value of attribute D is E (or that one value for the set-valued attribute D is E, resp.). Note that in F-Logic you may define $C[D \rightarrow E]$ and $C[D \twoheadrightarrow E]$, i.e. the same attribute id referred to by $\rightarrow$, and $\twoheadrightarrow$ virtually refer to different attributes, since the interpretations of $\rightarrow$ and $\twoheadrightarrow$ are separate in the F-Logic semantics. Note that in the context of our translation we only use $\twoheadrightarrow$ to denote attribute values.

An F-Logic formula is defined by combining molecules with the usual logic connectives $\wedge, \vee, \leftarrow, \leftrightarrow, \neg$ and quantifiers $\forall, \exists$.

Note that in F-Logic there is no distinction between classes and instances. An *object identifier* can denote a class, an instance, or an attribute, but there is no separation in σ for the identifiers denoting either. If required, such a separation can be enforced by explicit axiomatization in F-Logic or in a sorted version of F-Logic (cf. Section 17.1 of [Kifer et al., 1995]). The advantage of such an overloading object notion is that objects denote classes, instances and attributes depending on the syntactic context, which allows meta-statements that will be used in OWL Full$^-$, cf. Chapter 6.

As opposed to interpretation in classical logic, interpretations in F-Logic, apart from assigning a domain value to each element in the domain σ, additionally assign a partial function $I_{\twoheadrightarrow} : \sigma \rightarrow \mathcal{P}(\sigma)$ to each element of σ. This function interprets the attributes.

Besides a direct model-theoretic semantics, [Kifer et al., 1995] also identifies a Logic Programming-style (perfect-model)[4] semantics for F-Logic. There exist several implementations of this Logic Programming fragment of F-Logic, such as FLORA-2 [Yang et al., 2003a] and Ontobroker [Decker et al., 1999].

An *F program* (a collection of F-Logic formulas) is said to be well-typed if all data atoms implied by the program comply with the signatures implied by the program [Kifer et al., 1995]. The notion of type-correctness is not built into the logical language, but there is a separate logical meta-theory around signatures (using an extended syntax with the symbols $\Rightarrow$ and $\Rrightarrow$ in so-called signature molecules). This has two main advantages: (1) it is possible to use different theories for type-correctness for programs in the same language and (2) it enables checking where in the program typing errors occur, instead of just saying that the entire program (or Knowledge Base) is unsatisfiable (as in Description Logics). In our translation in Chapter 6 we do not use signatures but directly axiomatize typing in F-Logic.

[Kifer et al., 1995] provides a sound and complete proof theory for F-Logic, which is similar to resolution for classical (first-order) logic, but is extended with inference rules for (among others) subclassing and typing, and for other features.

[4]Please note that both Ontobroker and FLORA-2 implement Well-Founded Semantics (WFS) [Gelder et al., 1988], in order to allow non-stratified logic programs. For locally stratified logic programs, the WFS and Perfect-Model semantics coincide.

3 Drawbacks of OWL

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. At the same time, OWL has no notion of constraints. Furthermore, because the expressiveness of the $\mathcal{SHIF}$ (and especially $\mathcal{SHOIN}$) Description Logic language is beyond the capabilities of efficient rule-based engines, and because extending a Description Logic with Horn-like rules in a naive manner leads to undecidability issues [Levy and Rousset, 1998], one cannot easily extend OWL Lite/DL with a rule language, without running into undecidability problems; OWL Full is undecidable even without such rules.

OWL requires equality reasoning Equality is part of the OWL language. It is introduced through cardinality restrictions, functional properties, and explicitly asserted equality between individuals. Apart from doubts about the usefulness of this feature in the language, which we express below, we also doubt the computational tractability of reasoning with equality in the language. Although equality in the language does not add to the theoretical complexity of reasoning, there are practical implications of reasoning with equality. Namely, unification of terms requires complex satisfiability checks instead of simple syntactic matching, which can be done for a language without equality.

ABox reasoning in OWL is hard The satisfiability problem in $\mathcal{SHIF}$ has ExpTime complexity [1]; $\mathcal{SHOIN}$ has NExpTime complexity. Although this complexity refers to the combined complexity and the data complexity, on which the complexity of query answering depends, is P [Hustadt et al., 2004], current Description Logic implementations use complex Tableaux checks for query answering [Horrocks et al., 2000]. Reasons why Tableaux scales poorly for ABox reasoning are given in [Hustadt et al., 2004]. The major problem is that a Tableaux check is required for each individual in the knowledge base to see whether it is in the answer to the query. Although several optimizations exist [Haarslev and Möller, 2003; Haarslev and Möller, 2004], the fundamental problems are not solved. In our opinion it is impractical to require such complex reasoning for even the least expressive of the OWL species, since efficient ABox reasoning will play a major role on the Semantic Web. 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 is non-intuitive In OWL, 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. We illustrate this with an example:

Example 3.1. Assume the following OWL DL knowledge base:

[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.

```
ObjectProperty(hasPassenger domain(FlightSeat)
   range(Passenger))
Class(FlightSeat partial
   restriction(hasPassenger maxCardinality(1)))
Individual(seat1 type(FlightSeat)
   value(hasPassenger mary)
   value(hasPassenger john))
```

`FlightSeat` represents the seats in a particular flight. The property `hasPassenger` associates a seat in a flight with a passenger. A seat may only have one passenger, which is guaranteed by the restriction `maxCardinality(1)`. The individual `seat1` is asserted as instance of `FlightSeat`; `seat1` has two values for the property `hasPassenger`, namely `mary` and `john`.

□

From the above example the reasoner will draw the conclusion that `mary` and `john` 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 the `seat1` is overbooked, are not taken into account[2]. Issues become even more awkward when dealing with arbitrary cardinality restrictions: Assume that attibute p has a cardinality of 2 and three fillers are known `Individual(a type(C) value(p b) value(p c) value(p d))` Here, we have to infer a disjunction of equalities $b = c \vee b = d \vee c = d$, which is even worse than the situation in Example 3.1 above where the inference of equality was unique, i.e. all fillers of passengers booked for the same seat were inferred to refer to the same person. Apart from being hard to interpret by a common knowledge engineer, this kind of inference also has a high impact on the efficiency of underlying reasoning procedures. The unintuitive and ambiguous inference of equality is highly related to another limitation of OWL – the lack of constraints.

Deriving Class Membership through Value Restrictions In OWL it is possible to restrict the range of a property P to a class description C either through a range restriction in the property definition or through a local universal range restriction in a class definition. From this restriction it is inferred that every value of this property is a member of class C.

Example 3.2. Consider the OWL knowledge base from Example 3.1 extended with the following assertions:

```
Individual(seat3 type(FlightSeat))
Individual(seat2 type(FlightSeat)
   value(hasPassenger seat3))
```

The above assertions introduce two new instances of the class `FlightSeat` by the names of `seat2` and `seat3` plus a value for the property `hasPassenger` at `seat3`, namely `seat3`. □

From Example 3.2 we can infer that `seat3` is a `Passenger`. Clearly, there is some mistake in the individual assertions, because a seat cannot occupy another seat; only a passenger can. However, this mistake is not detected; instead the modeling mistake allows for additional (incorrect) inferences. Although it is possible in OWL DL to express disjointness of classes, in which case an inconsistency would be derived, we argue that in many application domains it

[2]Different to the semantics of OWL here, the authors of F-Logic [Kifer et al., 1995] tackle this issue in their language by allowing the user to specify which part of a program is allowed to explicitly infer equality and which part of the program would infer a modeling mistake [Kifer et al., 1995, Section 12.1].

is natural to assume disjointness of classes beforehand and only deviate from this assumption when classes are *known* not to be disjoint.

A direct way to model constraints is lacking in OWL. One would often intuitively expect that (domain, range and cardinality) restrictions in OWL correspond to constraints, which is not the case. Domain and range restrictions are used to *infer* new information about individuals, rather than to detect inconsistencies. Similar to the issue above, this is a general problem of the intent of OWL to use a pure first-order language for ontologies. Without non-monotonic features, modeling integrity constraints in the sense of database constraints is not possible: Cardinality restrictions in OWL are used to derive equality, as was shown in the description of the previous limitation. The only way to model value constraints, in the sense of database constraints, in OWL is by explicitly asserting the disjointness of the two classes[3] or enforcing the unique names assumption (UNA) by explicitly 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 $\mathbf{K}$. When using the $\mathbf{K}$ operator as a prefix to a concept ($\mathbf{K}C$) or role ($\mathbf{K}R$) 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 $\mathbf{K}$ 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 corresponding to an integrity constraint is then only generated, if an instance violating the integrity constraint is in the knowledge base. This corresponds with the usual notion of integrity constraints in deductive databases.

Difference in the treatment of abstract and concrete values OWL *does* use constraints when it comes to concrete values, or *literals*. Literals in OWL do adhere to the *unique name assumption* and thus do not require reasoning with equality, but merely checking of equality. Furthermore, universal or existential value restrictions are treated as constraints. Instead of *deriving* the type of a literal from a universal (allValuesFrom) value restriction or from a range restiction, the type of the literal is *checked* to see whether it corresponds with the type in the restriction. It has been argued that the difference in treatment of individuals and literals makes sense, because the domain of a data type is known. However, we argue that from the point of view of the user of the language, this conceptual distinction is not intuitive. Whereas axioms involving the abstract domain are used to *infer* (often contrary to the intuition) new knowledge, axioms involving the concrete domain are used to *check* whether the knowledge satisfies certain constraints.

Rule extension of OWL is not straightforward There have been several proposals for rule extensions for Description Logic languages. Three 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

[3]Disjointness can not be directly modeled in OWL Lite (it can be in OWL DL). 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 FlightSeat)`, which is equivalent to the following Description Logic statement: $\bot \equiv \mathit{Person} \sqcap \mathit{FlightSeat}$.

unary and binary predicates, respectively, in Horn clauses (e.g. $\mathcal{AL}$-log [Donini et al., 1998b] and CARIN [Levy and Rousset, 1998]), (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]) and (3) approaches which define a restricted interface between the Description Logic and the Logic Programming formalisms and allow for exchange of conclusions between the two (e.g. [Eiter et al., 2004]). The approaches under (1) and (2) are similar, however, the approaches under (1) do not allow predicates from the Description Logic knowledge base in the heads of the rules, whereas the approaches under (2) do, which means that in these approaches conclusions drawn from the rules can affect the Description Logic 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., 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 are not aware of any attempts to create a rule language based on OWL Full. However, such a language would certainly be undecidable, as OWL Full itself is already undecidable.

We argue that some of the above mentioned limitations can be overcome by using a more restricted form of OWL Lite and OWL DL, which can be translated into a Datalog program (without equality). This language can then be extended in a straightforward manner 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.

Inappropriate Layering Both the layering on top of RDFS and the layering of OWL species (especially the layering of OWL Full on top of OWL DL) is, in our opinion, inappropriate. The two smaller species of OWL, OWL Lite and OWL DL, are only layered on top of a restricted subset of RDFS, whereas the largest species of OWL, OWL Full, is completely syntactically and semantically layered on top of RDFS, cf. Figure 3.1.

Within OWL, the OWL DL species is neatly layered on top of OWL Lite. OWL DL removes several syntactical restrictions and adds several syntactical constructs. Semantically, OWL DL adds support for nominals and arbitrary cardinality restrictions (higher than 1) [Horrocks et al., 2003]. On the one hand, one could argue whether OWL Lite is really a sufficient restriction on OWL DL to justify creating a separate species. OWL Full, on the other hand, layers both on top of RDF(S) and OWL DL and because these languages are so different, a lot of tricks had to be employed to both syntactically and semantically layer

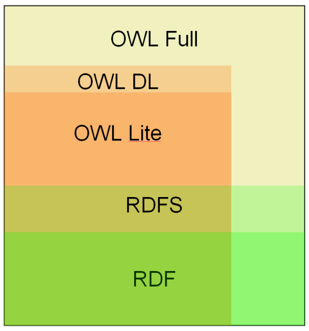

Figure 3.1: RDF(S)/OWL Layering

OWL Full on top of both languages.

In order to remedy the situation with the layering of OWL Full on top of both OWL DL and RDFS, two styles of semantics have been created for OWL DL. A direct model-theoretic semantics and an RDFS-based semantics [Patel-Schneider et al., 2004]. OWL Full, is unfortunately not properly semantically layered on top of OWL DL; entailment under OWL DL semantics is not equivalent to entailment under OWL Full semantics for the same ontology: OWL Full allows additional inferences. This discrepancy is caused by the incompatibility between the model-theoretic semantics of OWL DL and the axiomatic semantics of and syntactical freedom of RDFS. This raises doubts about the level of interoperability between the different species of OWL.

Limited Support for Datatypes OWL allows for a limited treatment of datatypes. In order to support datatypes, an OWL ontology is interpreted in two disjoint domains: the abstract domain and the concrete domain. An OWL reasoner deals only with the abstract domain and assumes a datatype oracle with a sound and complete procedure to decide 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. Negation of datatypes would be required for most Description Logic reasoners however, since during transformation to Negation Normal Form (NNF), which is performed by most DL reasoners, we would need to cope with negative datatype expressions not occurring negatively in the original DL expression.

- 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-than ($\geq$) relation for the `xsd:integer` domain in OWL. It is thus not possible to express, for example, that an adult is someone who is at least 18 years old. Note that such a statement is possible in many Description Logic extensions with concrete domains, e.g. in [Baader and Hanschke, 1991; Horrocks and Sattler, 2001] such an axiom can be expressed as $Adult \sqsubseteq \forall .hasAge. \geq_{18}$. Here, $\geq_{18}$ denotes all integers greater than or equal to 18 (which is a user-defined data type).

- OWL does not support user-defined datatypes. One would expect that OWL would support user-defined datatypes, because it uses the simple datatypes from XML Schema. In XML Schema it is possible for the user to define datatypes, but it is impossible to use standard datatypes in OWL, because there is no standard way to go from a URI reference to an XML Schema Datatype defined in an XML Schema Document [Patel-Schneider et al., 2004].

[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]).

In the remainder of this document we restrict the species of OWL, OWL Lite, OWL DL and OWL Full, to restricted subsets which overcome some of the problems mentioned in this chapter.

OWL Lite$^-$ is a subset OWL Lite that falls within the $\mathcal{DHL}$ Description Logic, which we described in Section 2.3.1. It enables efficient query answering using deductive databases and does not require complex reasoning with equality. Furthermore, because this language falls within the Horn fragment of First-order logic, rule extensions of the language are straightforward. Also, OWL Lite$^-$ does not allow to derive equality and, in order to stay within the Horn fragment, the language does not allow concrete values.

OWL DL$^-$ is an extension of OWL Lite$^-$ and a subset of OWL DL. It is the maximal subset of OWL DL that stays within $\mathcal{DHL}$ and thus within the Horn fragment. Therefore, OWL DL$^-$ does not increase complexity with respect to OWL Lite$^-$, but does include more features in order to give the modeler more flexibility when creating ontologies.

OWL Full$^-$ is a subset of OWL Full which extends OWL DL$^-$ by allowing one to treat the same identifier both as a class and an instance. This feature is beyond popular Description Logics. OWL Full$^-$ is semantically layered on top of OWL DL$^-$ and thus, OWL$^-$ overcomes the problems with inappropriate layering of the species of OWL.

Since we aim at staying within OWL here, we address the remaining drawbacks, namely the lack of a notion of constraints and limited support for datatypes, in the deliverable D20.3: OWL Flight [de Bruijn et al., 2004]. Thus, the present deliverable defines a strict subset of OWL upon which we will build a new, more useful ontology language in [de Bruijn et al., 2004].

4 OWL Lite$^-$

This chapter defines the ontology language OWL Lite$^-$, which is a proper subset of OWL Lite that can be translated to Datalog.

In this chapter, we restrict the syntax and semantics of OWL Lite to create OWL Lite$^-$ and discuss the features of OWL Lite that are not in OWL Lite$^-$.

In our investigation of the features of OWL Lite, we aim to find out which of these features make the language hard to reason with and which of the features make the language hard to model with. We aim to come up with a truly light-weight ontology language for the Semantic Web as a subset of the lightest species of OWL.

The chapter is further structured as follows: First, we define OWL Lite$^-$ and enumerate all of its features and also which features have been omitted. We provide the restrictions on the abstract syntax and the RDF syntax of OWL Lite, and define the semantics of OWL Lite$^-$ in terms of $\mathcal{DHL}$. A direct translation from OWL Lite$^-$ to Datalog follows from the results in Section 2.3.1. Finally, we describe the relationship between OWL Lite$^-$ and RDFS.

4.1 Defining OWL Lite$^-$

Table 4.1 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). We use 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 of the GCI is allowed to occur in the defined 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 defined part (the right-hand side), where the defined class coincides with one of the classes of the disjunction. An existential property restriction cannot 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 each feature of OWL Lite shown in Table 4.1 and explain why it is or is not included.

For our discussion of the particular 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

OWL Abstract Syntax	**DL syntax**	**OWL Lite$^-$**
Axioms		
`Class(`A `partial` C_1 `...` C_n`)`	$A \sqsubseteq C_i$	+ ($C_i \neq \bot$; $A \neq \top$)
`Class(`A `complete` C_1 `...` C_n`)`	$A \equiv C_1 \sqcap \ldots \sqcap C_n$	+ ($A, C_i \neq \bot, \top$)
`EquivalentClasses(`A_1 `...` A_n`)`	$A_1 \equiv \ldots \equiv A_n$	+ ($A_i \neq \bot, \top$)
`ObjectProperty(`R `super(`R_1`)...super(`R_n`)`	$R \sqsubseteq R_i$	+
`domain(`A_1`) ... domain(`A_n`)`	$\top \sqsubseteq \forall R^-.A_i$	+ ($A_i \neq \bot$)
`range(`A_1`) ... range(`A_n`)`	$\top \sqsubseteq \forall R.A_i$	+ ($A_i \neq \bot$)
`[inverseOf(`R_0`)]`	$R \equiv R_0^-$	+
`[Symmetric]`	$R \equiv R^-$	+
`[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(`A_1`) ... type(`A_n`)`	$o \in A_i$	+
`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(`A`))`	$\exists R.A$	–
`restriction(`R `allValuesFrom(`A`))`	$\forall R.A$	partial* ($A \neq \bot$)
`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 4.1: Features of OWL Lite present in OWL Lite$^-$

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_0 roughly corresponds to the expressivity of $\mathcal{DHL}$.
- 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 and an inconsistency in the program is derived. Note that if a Datalog engine does not explicitly allow the use of integrity constraints, these can be easily emulated by introducing rules with a special predicate symbol *ic* in the head for each integrity constraint. If the extension of the *ic* predicate is non-empty, this means the integrity constraint is violated. LP_0 roughly corresponds to the expressivity of $\mathcal{DHL}_\bot$.
- LP_2: Datalog(IC,=), which additionally allows equality in rule heads. Note that equality in Datalog significantly increases the complexity of the implementation and query answering with equality can be expected to be harder[1].
- LP_3: Prolog(IC,=), which generalizes LP_2 by allowing function symbols and unsafe rules. Note that we do not allow negation.

Note 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 $\mathcal{LP}_3$ in Volz

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

[Volz, 2004]. In his layering, Volz 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 than integrity constraints.

We now discuss every feature of OWL Lite separately. The discussion of resulting complexity impacts for the support of particular features of the language is 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 defined part) and `owl:Thing` ($\top$) is not allowed to occur on the left hand side (the defining 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 allowing integrity constraints, basically stating 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 the bottom concept `owl:Nothing` amounts to an integrity constraint, it is expressible in LP_1.

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. Complete class definition with `owl:Nothing` on the left hand side can be used to emulate the `DisjointClasses(`$C_1 \ldots C_n$`)` statement in OWL DL, by defining equivalency with `owl:Nothing` for all pairs C_i, C_j.

Stating disjointness of classes is useful in cases such as the Example 3.2 from Chapter 3. However, when treating a range restriction as a "real" constraint[3], this would not be necessary, because the constraint would be violated if `seat3` could not be inferred to be a member of `Passenger`.

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`. On the right-hand side `owl:Thing` 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.

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

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

[2]A remark is 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.

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

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 are in OWL Lite$^-$.

Functional and Inverse Functional properties (Inverse) functional properties cannot be expressed in OWL Lite$^-$ since functional properties require equality in the language, which is not part of Datalog; Note that (inverse) functionality of properties is expressible in LP_2 by equality.

(Inverse) Functionality of a property is equivalent to a maximal cardinality constraint of 1. As we have argued in Chapter 3, the maximal cardinality restrictions in OWL Lite cause derivation of equality in a non-intuitive manner. From this perspective, the lack of functional 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 are no functional 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 Property values (role fillers) can be expressed in OWL Lite$^-$.

(In)equality among individuals 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. Thus, individual (in)equality can only be expressed in LP_2.

On the one hand, equality support is required for working with the web. For example, distinct URIs can point to the same resource. Therefore, in a web context we need to be able to deal with equality. On the other hand, support for equality in OWL is inadequate (cf. [Fensel, 2003, pp. 45]). First, one must list all equalities explicitly, which may take the modeler prohibitively long to do. Second, equalities can be inferred through cardinality constraints. As explained in the earlier examples such inferences can generate unintended equalities and can be computationally expensive. In a nutshell, the equality inference of OWL is (1) dangerous, (2) inadequate, and (3) costly without added value in return. It is dangerous because innocuous mmodeling mistakes can lead to inferences of even more erroneous facts. It is inadequate because none of the required equality mechanisms for literals and URIs can be expressed. 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 $\top$ (see Table 2.1). As we have discussed 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 a special symbol such as *true* to mimic the $\top$ concept.

Therefore, in general `owl:Thing` is not allowed in OWL Lite$^-$ except in special cases, where it can be eliminated. Also, it is easy to extend any Datalog implementation with support for the *true* symbol.

owl:Nothing `owl:Nothing` is translated to the symbol $\bot$ (see Table 2.1). As we have discussed 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 a special symbol such as $false$ to mimic the empty concept $\bot$.

Therefore, in general `owl:Nothing` is not allowed in OWL Lite$^-$, except in special cases, where it can be eliminated. Also, it is easy to extend Datalog implementations with support for the $false$ symbol, in the head of rules: In general, use of the $false$ symbol in the head of rules 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 existentially quantified variables can be eliminated through skolemization, i.e. by substituting it with a skolem function. However, function symbols are not allowed in Datalog and thus neither is skolemization.

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 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 size of the data [Dantsin et al., 2001]. We must note here that this additional complexity is only introduced when you actually use the disjunction in the head. Without disjunction in the head, data complexity is still in P. Thus, one only pays the performance penalty when actually using the additional expressiveness. However, if the additional expressiveness is given, people are more likely to abuse it.

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 of the GCI, the axiom can no longer be translated to plain Datalog. Therefore, we disallow the use of the minimal cardinality restrictions 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 A minimal cardinality restriction of 1 cannot 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 equivalent to a universal value restriction with `owl:Nothing` as its range. As we have seen above, this cannot be expressed in OWL Lite$^-$. However, it would be possible to express a maximal cardinality of 0 using a partial class definition in LP_1.

The usefulness of a maximal cardinality restriction of 0 is arguable. Why define a property when without any property fillers? The only use of this facility is is to say that a propertay *may not* be used with a particular class. 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 3, the cardinality restrictions in OWL Lite lead to the non-intuitive deduction of equality. We envisage that future extensions of OWL Lite$^-$ will introduce database-style cardinality constraints.

To sum up, 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 4.1 and restricting the use of the `allValuesFrom` property restriction, i.e. restricting the language constructs of OWL Lite to $\mathcal{DHL}$ (or LP_0, respectively).

4.2 OWL Lite$^-$ RDF Syntax

Table 4.1 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]:

Definition 4.1. *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].

4.3 OWL Lite$^-$ Model-Theoretic Semantics

The features indicated in Table 4.1 show that OWL Lite$^-$ is in pure $\mathcal{DHL}$ (see Section 2.3.1) and therefore inherits the model-theoretic Description Logic semantics [Baader et al., 2003] of that language. Furthermore, because $\mathcal{DHL}$ is in the function-free Horn fragment of First-Order Logic, efficient query answering can be done with deductive database.

[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]

4.4 Transforming OWL Lite$^-$ to Datalog

Since all axiom of OWL Lite$^-$ directly translate to $\mathcal{DHL}$, we can immediately use the mapping outlined in Table 2.1 in Section 2.3.1 to translate OWL Lite$^-$ to Datalog.

4.5 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 explain the relationship between OWL Lite$^-$ and RDFS.

We first describe the features of RDFS, which are in OWL Lite$^-$. Then, we describe the restrictions that OWL Lite$^-$ imposes on RDFS. Finally, we describe the expressivity which OWL Lite$^-$ adds on top of (the OWL Lite restricted form 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.

OWL Lite is syntactically and semantically layered on top of a subset of RDFS only. We will show that this RDFS subset of OWL Lite is the same as the RDFS subset of OWL Lite$^-$.

4.5.1 RDFS in OWL Lite$^-$

It turns out that the RDFS subset of OWL Lite$^-$ corresponds to 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 4.1). 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 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 of properties 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$^-$ and OWL Lite.

4.5.2 Restrictions on RDFS imposed by OWL Lite$^-$

OWL Lite, and thus OWL Lite$^-$, imposes 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 4.2.
- There are general restrictions on the RDF graph in the context of serializing OWL into RDF, because, the modeling of OWL in RDF consumes several triples that must occur together in a particular form in the graph, having somehow the semantics encoded syntactically, which we consider awkward.

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 as 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 on the Semantic Web.

4.5.3 Expressivity of OWL Lite$^-$ compared to RDFS

Since OWL Lite$^-$ is a restricted subset of OWL Lite, it is useful to see what OWL Lite$^-$ covers compared to the intersection of RDFS and OWL Lite.

The features of OWL Lite$^-$ that are not in RDFS are:

- Complete (i.e. necessary and sufficient) class definitions, whereas RDFS only allows partial (i.e. necessary) class definitions.
- Equivalence among classes and equivalence among 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$^-$ (analogously for subproperties).

[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.

- 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.

4.6 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 these features not to be included in OWL Lite$^-$.

For the definition 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 notably 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 to a large extent.

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 seen that by falling into $\mathcal{DHL}$, there is a starightforwar translation from OWL Lite$^-$ to Datalog. Furthermore, we have discussed the relationship between OWL Lite$^-$ and RDFS and have shown that OWL Lite$^-$ is a proper extension of RDFS restricted to OWL (Lite). We have furthermore shown that OWL Lite$^-$ adds significant expressive power on top of this fragment 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 either 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 `value(`o`)` property restriction ($\exists R.\{o\}$).

5 OWL DL$^-$

In the previous chapter we introduced OWL Lite$^-$ as the subset of OWL Lite, which can be evaluated on a Datalog engine. In this chapter, we extend OWL Lite$^-$ to OWL DL$^-$, by including the `value` restriction and allowing nested restrictions, which can also be translated and evaluated using Datalog engines. We allow the `value` restriction in both partial and complete class definitions.

More specifically, OWL DL$^-$ extends OWL Lite$^-$ in the following ways:

Re-introducing the `value` restriction The $\mathcal{SHIF}$ Description Logic underlying OWL Lite does not allow the use of individuals in concept descriptions. Therefore, the `value` property restriction, which is in OWL DL, is not in OWL Lite. However, as was shown by [Volz, 2004], the `value` restriction does not introduce additional complexity when performing query answering in deductive databases. Therefore, we re-introduce this restriction.

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 4.1, this translates to an intersection statement in Description Logic syntax.

With OWL DL$^-$ we aim to capture the maximal subset of OWL DL which falls inside $\mathcal{DHL}$ and thus the Horn fragment. We abandoned of some of the syntactical restrictions imposed by OWL Lite$^-$, because many of these restrictions do not really restrict the expressiveness, but rather restrict the user in ways to write down certain axioms.

OWL Lite$^-$ provides basic conceptual modeling support through classes, class hierarchies, properties, property hierarchies, local property restrictions and several property characteristics. OWL DL$^-$ adds to OWL Lite$^-$ support for more complex axiom definitions. The user can write down arbitrary $\mathcal{DHL}$ axioms and is not restricted to the rigid syntactical structure of OWL Lite$^-$.

The chapter is further structured as follows: We define OWL DL$^-$ and enumerate all the features from OWL DL which are present or have been omitted. We then provide a translation from OWL DL$^-$ to Datalog.

5.1 OWL DL$^-$ Abstract Syntax

We begin with the description of differences between OWL DL and OWL Lite. After that, we outline the differences between OWL Lite$^-$ and OWL DL$^-$.

OWL DL adds a number of features on top of OWL Lite and also removes some restrictions:

- Property domain and range restrictions allow descriptions in place of named classes.

OWL Abstract Syntax	**DL syntax**	**OWL DL$^-$**
Axioms		
Class(A partial C_1 ... C_n)	$A \sqsubseteq C_i$	$+$ ($C_i \neq \bot$; $A \neq \top$)
Class(A complete C_1 ... C_n)	$A \equiv C_1 \sqcap \ldots \sqcap C_n$	$+$ ($A, C_i \neq \bot, \top$)
EnumeratedClass(A o_1 ... o_n)	$A \equiv \{o_1, \ldots o_n\}$	$-$
SubClassOf(C_1 C_2)	$C_1 \sqsubseteq C_2$	$+$
EquivalentClasses(C_1 ... C_n)	$C_1 \equiv \ldots \equiv C_n$	$+$ ($C_i \neq \bot, \top$)
DisjointClasses(C_1 ... C_n)	$C_i \sqcap C_j \sqsubseteq \bot$	$-$
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]	$R \equiv R^-$	$+$
[Functional]	$\top \sqsubseteq \leqslant 1R$	$-$
[InverseFunctional]	$\top \sqsubseteq \leqslant 1R^-$	$-$
[Transitive])	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	$\bot$	$-$
intersectionOf(C_1 ... C_n)	$C_1 \sqcap \ldots \sqcap C_n$	$+$
unionOf(C_1 ... C_n)	$C_1 \sqcup \ldots \sqcup C_n$	lhs*
complementOf(C_0	$\neg C_0$	$-$
oneOf(o_1 ... o_n)	$\{o_1, \ldots o_n\}$	lhs*
restriction(R someValuesFrom(C))	$\exists R.D$	lhs* ($D \neq \bot$)
restriction(R allValuesFrom(C))	$\forall R.D$	rhs** ($D \neq \bot$)
restriction(R value(o))	$\exists R.o$	$+$
restriction(R minCardinality(1))	$\geqslant 1R$	lhs*
restriction(R minCardinality(n)) ($n > 1$)	$\geqslant nR$	$-$
restriction(R maxCardinality(n))	$\leqslant nR$	$-$

* May only be used on the left-hand side (as the first argument) of `SubClassOf`
** May only be used in partial class definitions and on the right-hand side (second argument) of `SubClassOf`

Table 5.1: Features of OWL DL present in OWL DL$^-$

- Cardinality restrictions are no longer limited to 0 and 1.
- Descriptions can contain **nested** universal and existential restrictions, i.e. in the `allValuesFrom` and the `someValuesFrom` restrictions, descriptions are allowed in place of named classes.
- OWL DL extends the list of descriptions of OWL Lite (consisting of Class IDs and restrictions) to include `unionOf`, `intersectionOf`, `complementOf` and `oneOf` thereby allowing complex class definitions. Note that the only description which adds expressiveness is `oneOf`. The other descriptions can all be encoded using existing constructors in OWL Lite (cf. [Horrocks et al., 2003]).
- OWL DL add the following constructors for class axioms: `Enumerated-Class`, `DisjointClasses` and `SubClassOf`. Note that the only constructor which adds expressiveness is `EnumeratedClass`.

The difference between OWL Lite and OWL DL is mainly syntactic sugar. The only features added which increase the expressiveness are the support for nominals and the support for arbitrary number restrictions.

Table 5.1 shows the features of OWL DL, which are present in OWL DL$^-$, based on [Volz, 2004, Table 4.7, pp. 111]. For the complete abstract syntax of OWL DL$^-$, see Appendix A. The differences between OWL Lite$^-$ and OWL DL$^-$ are:

- We allow the `intersectionOf` description in place of named classes (i.e. in class definitions and in value restrictions).
- We allow the `value` restriction in both partial and complete class definitions.
- We allow the `SubClassOf` construct, which explicitly specifies a subsumption relationship between two descriptions. Note that within the `SubClassOf` construct, arbitrary descriptions are allowed. Note also that descriptions that are only allowed to occur in partial class definitions, are not allowed to occur on the left-hand side, which is the first argument of the `SubClassOf` construct. Note also that this first argument is the only place the `unionOf` construct and the existential value restriction `someValuesFrom` are allowed to occur.
- We add the `unionOf` construct, which may occur as the first argument of `SubClassOf`.
- We also allow `someValuesFrom` to occur as the first argument of `SubClassOf`.
- Unlike [Volz, 2004], we do allow the `oneOf` construct to occur on the left-hand side of GCI, i.e. as the first argument of `SubClassOf`. This follows directly from Corollary 2.1 in Section 2.3.1.

5.2 OWL DL$^-$ Semantics

As indicated by the features shown in Table 5.1 OWL DL$^-$ is in pure $\mathcal{DHL}$ (see Section 2.3.1) and therefore inherits the model-theoretic Description Logic semantics [Baader et al., 2003] of the language. Furthermore, because $\mathcal{DHL}$ is in the function-free Horn fragment of First-Order Logic, efficient query answering can be done with deductive database.

5.3 Transforming OWL DL$^-$ to Datalog

Since all axioms of OWL DL$^-$ directly translate to $\mathcal{DHL}$, we can immediately use the mapping outlined in Table 2.1 in Section 2.3.1 to translate OWL Lite$^-$ to Datalog.

5.4 Limitations of OWL DL$^-$

OWL DL$^-$ adds limited expressiveness to OWL Lite$^-$. More specifically, the language adds the following features introducing additional expressiveness:

- The `value` restriction is added, indicating default property fillers.

- The `someValuesFrom` restriction and the `oneOf` description are added, albeit they are only allowed to occur as the first argument of `SubClassOf`.

Many of the limitations of OWL Lite$^-$ remain, such as the lack of datatypes, unsatisfactory layering on top of RDF(S) and the inability to define meta-classes. Concerning the layering on top of RDF(S) the same remarks apply as for OWL Lite$^-$ in Section 4.5.2. Some of these limitations are overcome in OWL Full$^-$, which is the subject of the next chapter.

Other limitations, such as the lack of a notion of constraints and the lack of datatypes, still remain and we address these issues in another deliverable [de Bruijn et al., 2004].

6 OWL-Full$^-$

This chapter introduces OWL Full$^-$, an extension of OWL DL$^-$ with a meta-class facility. Note that OWL Full$^-$ is conceptually not defined via a restriction of OWL Full, as opposed to OWL Lite$^-$ and OWL DL$^-$, which are defined as restrictions of OWL Lite and OWL DL, respectively. Instead, OWL Full$^-$ is defined as an extension of OWL DL$^-$ in the direction of OWL Full allowing a limited meta-class facility. The rationale behind this is that we do not want to cover full RDF(S) with its awkward circular dependencies within the rdf vocabulary itself but only enable meta-class facilities outside the RDF and OWL vocabulary, cf. Figure 6.1. This allows us to include a restricted subset of RDF in OWL Full$^-$ which is not available in OWL DL.

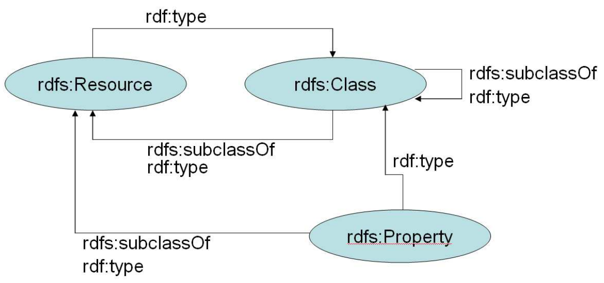

Figure 6.1: Subclass and type relations among the RDF(S) vocabulary.

We define the semantics of OWL Full$^-$ through an axiomatization of the OWL DL$^-$ Abstract Syntax in F-Logic. We then relax the condition on separation of the vocabulary. More specifically, V_C and V_I, the sets of class and individual names, respectively, are no longer required to be disjoint. However, we still impose some restrictions on the use of OWL and RDFS keywords.

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. Since OWL Full is an extension of RDFS, this feature is also available in OWL Full. However, OWL Lite and OWL DL do not properly layer on top of RDFS, as several features of RDFS are not available in OWL Lite and DL. There are several properties of RDFS which complicate the layering of a Description Logic-style language on top of RDFS (cf. [Horrocks and Patel-Schneider, 2003b; Horrocks et al., 2003]):

1. RDF has a triple-based data model. In RDF, it is not possible to restrict combinations of triples or to require certain triples to occur together. Therefore, only simple binary predicates can be expressed in RDF and, it is not possible to express more complex descriptions in RDF. For example, a local property restriction in OWL is conceptually a ternary predicate with its arguments being the class to which it is applied, the property, and the value for the restriction – which can be a number for a cardinality restriction or a description for a value restriction. Obviously, such a ternary predicate can be expressed using a number of binary predicates, but we believe that such a serialization of OWL constructs into triples is flawed: In RDF it is not possible to require the triples corresponding to a certain OWL construct to co-occur. OWL DL solves this problem by only allowing certain patterns of RDF triples to occur, through a mapping

from the abstract syntax to RDF triples [Patel-Schneider et al., 2004], but this seems to be feasible only in one direction whereas extraction of the actual OWL semantics from a general corresponding RDF graph in turn is non-trivial.

2. RDF(S) meta-predicates, which are used, for example, for creating classes (rdfs:Class), subclassing (rdfs:SubClassOf) and typing instances (rdf:Type) are all part of the domain of interpretation. Each resource in the domain of resources (IR) can occur as either the subject or object of any triple. A special type of resources, called predicates (domain IP, which is a subset of IR) can occur as the predicate of any triple. Since they are also resources, the predicates can also occur as the subject or object of a triple. OWL DL restricts RDF(S) by not allowing the use of language constructs in places where they would cause problems for the Description Logic.

3. In RDF(S) there is no distinction between the types of resources. There is no separation between the resources denoting classes, properties and individuals, respectively. Description Logics does make this separation. A class corresponds to a unary predicate, a property corresponds to a binary predicate and an individual corresponds to a constant. In a standard First-Order framework (Description Logics is a subset of the First-Order logic), it is not possible to use the same name for both a class and an individual. OWL DL strictly separates the vocabulary for classes, properties and individuals. An identifier cannot be both a class and an instance or both a class and a property.

There is a strong expectation that treatment of classes as instances will often occur on the Semantic Web (cf. [Schreiber, 2002]). For that reason, we extend OWL DL$^-$ with a meta-class facility based on F-Logic [Kifer et al., 1995].

This chapter is further structured as follows. We first analyze the use of meta-modeling and the interaction with logical languages. Then, we describe several possible underlying formalisms for meta-modeling, after which we provide a translation of OWL DL$^-$ to F-Logic. We then describe the OWL Full$^-$ abstract syntax and semantics. Finally, we describe the major limitations of OWL Full$^-$.

6.1 Meta-modeling

As we have outlined in Section 4.5.2 already, there are three main properties of RDF which complicate layering of a Description Logic-style language on top of it:

- The triple-based data model is inadequate for more expressive ontology languages whose semantics does not easily map onto the simple graph model of RDF(S).
- Inclusion of RDF(S) and OWL vocabulary within RDF(S) itself, cf. Figure 6.1.
- Strict separation of the vocabulary between individuals, classes and properties.

Clearly the first point about the limitations of the triple-based data model when it comes to more expressive language constructs cannot be overcome if one wants to fully layer a more expressive language on top of RDFS. Therefore, we believe that the chosen approach for the mapping of OWL DL to RDF triples [Patel-Schneider et al., 2004] is the right way to go and we will follow this approach.

With respect to the second point regarding language constructs in the domain of interpretation, we also follow the approach of OWL DL and do not allow language constructs to occur in place of individuals, classes or properties. The arbitrary use of language constructs in an ontology can easily lead to undecidability (cf. [Hustadt et al., 2004, Chapter 9]) and does not seem to serve any useful purpose.

With respect to the third point, the separation of vocabularies for class, properties and individuals, we do not fully agree with this separation introduced by OWL DL. It is often useful and conceptually necessary to treat the same resource as both a class and an individual (cf. [Schreiber, 2002; Noy, 2004]).

There have been proposals to circumvent this separation in certain ways outside the language (cf. [Noy, 2004]), however, the introduction of this separation of classes, properties and instances is artificial and inconvenient with respect to modeling support.

[Hustadt et al., 2004, Chapter 9] proposes a semantics for meta-modeling as an extension of a Description Logic. This semantics is based on HiLog [Chen et al., 1993], which solves the problem of the meta-modeling by assigning an interpretation as individual, as class and as property to each identifier. [Hustadt et al., 2004, Chapter 9] also shows that meta-modeling only adds expressiveness if equality between resources can be derived in the language. Since OWL DL$^-$ does not allow for the deriving of equality, added meta-modeling would not increase the expressiveness of the language. Therefore, it can be argued that meta-modeling should be handled on a meta-logical level instead of in the logic itself. It is always clear from the context whether an identifier denotes a class, property or individual. Therefore, each individual can be uniquely renamed (e.g. by appending '_c' to each class, '_p' to each property and '_i' to each individual) before reasoning with the ontology. This is in fact a syntactical trick to allow for meta-modeling in the language. In a sense, HiLog also employs a syntactical trick by interpreting each identifier in different ways.

6.2 Classes-as-instances in OWL Full$^-$

In this section we explore ways of adding meta-modeling to OWL DL$^-$. We see three styles of semantics which could be used for the meta-class facility in OWL Full$^-$. We refer to these styles as the RDFS(FA) style, the HiLog style and the F-Logic style.

RDFS(FA) style meta-classes RDFS(FA) [Pan and Horrocks, 2003a; Pan and Horrocks, 2003b] specifies a semantics for RDFS which facilitates easy layering of Description Logic-based languages (e.g. OWL Lite/DL) on top of RDFS. The issue that the same resource in RDFS can be both a class and an instance, and the issue that language constructs are part of the same domain of interpretation as the user-defined resources, are resolved by defining strata for these resources.

RDFS(FA) divides the interpretation of an ontology into layers, or *strata*. The names in these strata are disjoint, i.e. a resource occurring in one stratum cannot occur in a different stratum. Furthermore, a resource occurring in a particular stratum always refers to a set of resources 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 term corresponds to an instance of a class in the stratum directly above it and to a set (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.

Because the strata are strictly separated, the same resource can never be an instance of itself and, when reasoning with two adjacent strata at a time, standard Description Logic reasoning can be used.

A limitation is that we cannot treat a resource as both an instance and an object at the same time, but for many reasoning scenarios this is not a problem.

Extending OWL Lite$^-$ with RDFS(FA) requires some pre-processing when reasoning with a standard Datalog implementation. A Datalog implementation which is extended with HiLog, such as FLORA-2 [Yang et al., 2003b], could be readily used to handle such an extension.

HiLog style meta-classes Because of its higher-order syntax, HiLog [Chen et al., 1993] can be easily used to extend a Description Logic-based language, such as OWL Lite$^-$, to include meta-classes. A limitation is that a formula in first-order logic (and thus Description Logic) and the same formula in HiLog do not always have the same semantics. The semantics of FOL and HiLog (for a FOL formula) only coincide if the formula is cardinal, i.e. if the cardinality of the domain is at least as high as the number of symbols used in the language. This is the case for any set of equality-free sentences [Chen et al., 1993, pp. 13, 14, Lemma 3.2, 3.3] and thus applying HiLog semantics to OWL Lite$^-$ does not change the OWL Lite$^-$ semantics. Therefore, HiLog could easily be used to provide a meta-class facility for OWL Lite$^-$ and also for OWL Full$^-$, as long as no equality is introduced in the language.

However, using HiLog in the proposed way does not overcome the problem of having RDFS constructs in the domain of interpretation. Therefore, in order to provide a clean layering on top of RDFS, a new semantics for RDFS is necessary, which differs both from the semantics proposed by Hayes [Hayes, 2004] and the semantics proposed by Pan and Horrocks [Pan and Horrocks, 2003a]

F-Logic style meta-classes When choosing an F-Logic [Kifer et al., 1995] style meta-modeling facility, the semantics of the language needs to be related to F-Logic. In F-Logic a class is not represented by a predicate, but by an object, which points to a set of objects. This provides maximal flexibility in combining classes and instances, allowing a class to be defined as an instance of itself. Actually, the semantics of F-Logic style meta-classes is not far from HiLog-style meta-classes. In both languages, an object can be treated as a class (or unary predicate in HiLog) or as an individual, which can be an instance of other classes (in the extension of a unary predicate in HiLog). The main difference between HiLog and F-Logic is that F-Logic has frame-based constructs to represent classes and properties. It is therefore a more suitable language to represent classes, properties, and the like.

It is known that Description Logic languages can be axiomatized in F-Logic [Balaban, 1995]. However, this is based on the first-order style semantics for F-Logic originally specified in [Kifer et al., 1995]. Current implementations of F-Logic, such as FLORA-2 [Yang et al., 2003b] and OntoBroker [Decker et al., 1999], use a Logic Programming-style semantics. It is not known how far Description Logics can be translated into this style of F-Logic. However, there does exist a full semantics-preserving translation from OWL DL$^-$ to Datalog (cf. Section 5.3, which is a subset of the current Logic Programming-style implementations of F-Logic. Therefore, OWL DL$^-$ can be axiomatized in F-Logic (LP), as well as OWL Full$^-$, as long as it stays within the expressiveness of currently implemented logic programming formalisms. We will provide such a translation from OWL DL$^-$ to the Horn fragment of F-Logic in the next section.

Our rationale for using F-Logic as the basis for the meta-classes facility is that it provides a frame-based syntax and has the notions of classes attributes and instances, whereas HiLog relies on predicates and extensions of predicates. Equality is not a feature of OWL Full$^-$ and thus meta-modeling does not add expressiveness to the language (cf. [Hustadt et al., 2004, Chapter 9]). However, on a conceptual level, meta-modeling needs to be supported by the language and F-Logic provides a means for this.

6.3 From OWL DL$^-$ to Frame Logic

It is conjectured but not proven, that there exists a translation from OWL Full to First-Order Logic (FOL), cf. Footnote 2 on page 4 (also [Hustadt et al., 2004, Chapter 9]). In this chapter we shall not attempt to translate OWL Full to First-Order Logic. We do not define OWL Full$^-$ as a restriction of OWL Full, as was done in the previous chapters where OWL Lite$^-$ and OWL DL$^-$ were defined as restrictions of OWL Lite and OWL DL, respectively. Instead, we define OWL Full$^-$ as an extension of OWL DL$^-$ *towards* OWL Full, but not including full freedom of RDF(S). The main rationale for this is that we only want to support meta-modelling for resources outside the RDF(S) and OWL vocabulary but not the inclusion of the language vocabulary in the interpretation itself.

In order to do that, we first translate OWL DL$^-$ to Frame Logic [Kifer et al., 1995], which provides a higher-order syntax, but semantically stays inside First-Order Logic. We then eliminate some of the restrictions imposed by OWL DL on the use of the vocabulary and use the translation to Frame Logic as the semantic basis for this extension. Note that this takes us out of the Description Logic world.

In order to reformulate OWL DL$^-$ in terms of F-Logic, we build upon the translations from Description Logics to FOL based on [Borgida, 1996] and for translating Description Logics to F-Logic based on [Balaban, 1995].

Since FOL is a syntactical subset of F-Logic we could use the translation from [Borgida, 1996] directly, but here we want to come up with a more direct translation, using the features of F-Logic. This saves us from having to axiomatize subclass relationship, etc. Furthermore, F-Logic has a convenient and intuitive frame-based syntax, which allows us to define future extensions based on this translation more easily.

Table 6.1 presents the axiomatization of OWL DL$^-$ in F-Logic by means of a recursive translation function tr. In the Table we use the following notions: A refers to named classes, C refers to descriptions, and R is used to describe properties. x, y, z are variables in F-Logic, and X is a meta variable to be substituted with the actual variable during the translation. Note that only ontologies which are valid with respect to the OWL DL- abstract syntax can be translated.

In order to translate class axioms, the translation function $tr()$ takes an F-Logic expression from the translation of the left-hand side, $tr_{lhs}()$, of a subclass relationship (or partial/complete class descriptions, respectively) as the first argument, the right-hand side of a subclass relationship as the second argument,

OWL DL$^-$ Abstract Syntax	**F-Logic**
Mapping OWL DL$^-$ Class Axioms to F-Logic	
$\texttt{Class(}A\ \texttt{partial}\ C_1\ \texttt{...}\ C_n\texttt{)}$	$\bigwedge tr(x_{new} : A, C_i, x_{new})$
$\texttt{Class(}A\ \texttt{complete}\ C_1\ \texttt{...}\ C_n\texttt{)}$	$\bigwedge tr(x_{new} : A, C_i, x_{new}) \wedge$ $tr(\bigwedge tr_{lhs}(C_i, x_{new}), A, x_{new})$
$\texttt{EquivalentClasses(}C_1\ \texttt{...}\ C_n\texttt{)}$	$\bigwedge_{i \neq j} tr(tr_{lhs}(C_i, x_{new}), C_j, x_{new})$
$\texttt{SubClassOf(}C_1\ C_2\texttt{)}$	$tr(tr_{lhs}(C_1, x_{new}), C_2, x_{new})$
Mapping OWL DL$^-$ left-hand side Descriptions to F-Logic	
$tr_{lhs}(A, X)$	$X : A$
$tr_{lhs}(\texttt{intersectionOf(}C_1\ \texttt{...}\ C_n\texttt{)}, X)$	$\bigwedge(tr_{lhs}(C_i, X))$
$tr_{lhs}(\texttt{unionOf(}C_1\ \texttt{...}\ C_n\texttt{)}, X)$	$\bigvee(tr_{lhs}(C_i, X))$
$tr_{lhs}(\texttt{restriction(}R\ \texttt{value(}o\texttt{))}, X)$	$X[R \twoheadrightarrow o]$
$tr_{lhs}(\texttt{restriction(}R\ \texttt{someValuesFrom}\ C\texttt{)}, X)$	$X[R \twoheadrightarrow x_{new}] \wedge tr_{lhs}(C, x_{new})$
$tr_{lhs}(\texttt{oneOf(}o_1, \ldots, o_n\texttt{)}, X)$	$\bigvee X = o_i$
$tr_{lhs}(\texttt{restriction(}R\ \texttt{minCardinality(1))}, X)$	$X[R \twoheadrightarrow x_{new}]$
Mapping OWL DL$^-$ right-hand side Descriptions to F-Logic	
$tr(Expr, \texttt{intersectionOf(}C_1\ \texttt{...}\ C_n\texttt{)}, X)$	$\bigwedge tr(Expr, C_i, X)$
$tr(X : A_1, A_2, X)$	$A_1 :: A_2$
$tr(Expr, A, X)$	$Expr \rightarrow X : A$
$tr(Expr, \texttt{restriction(}R\ \texttt{value(}o\texttt{))}, X)$	$Expr \rightarrow X[R \twoheadrightarrow o]$
$tr(Expr, \texttt{restriction(}R\ \texttt{allValuesFrom}\ C\texttt{)}, X)$	$tr(Expr \wedge X[R \twoheadrightarrow x_{new}], C, x_{new})$
Mapping OWL DL$^-$ Property Axioms to F-Logic	
$\texttt{ObjectProperty(}R\ \texttt{[super(}R_1\texttt{)...super(}R_l\texttt{)]}$	$\bigwedge x[R \twoheadrightarrow y] \rightarrow x[R_i \twoheadrightarrow y]$
$\texttt{[domain(}C_1\texttt{)}\ \texttt{...}\ \texttt{domain(}C_m\texttt{)]}$	$\bigwedge tr(x[R \twoheadrightarrow y], C_i, x)$
$\texttt{[range(}C_1\texttt{)}\ \texttt{...}\ \texttt{range(}C_n\texttt{)]}$	$\bigwedge tr(x[R \twoheadrightarrow y], C_i, y)$
$\texttt{[inverseOf(}R'\texttt{)]}$	$x[R \twoheadrightarrow y] \rightarrow y[R' \twoheadrightarrow x] \wedge$ $x[R' \twoheadrightarrow y] \rightarrow y[R \twoheadrightarrow x]$
$\texttt{[Symmetric]}$	$x[R \twoheadrightarrow y] \rightarrow y[R \twoheadrightarrow x]$
$\texttt{[Transitive])}$	$x[R \twoheadrightarrow y] \wedge y[R \twoheadrightarrow z] \rightarrow x[R \twoheadrightarrow z]$
$\texttt{SubPropertyOf(}R_1\ R_2\texttt{)}$	$x[R_1 \twoheadrightarrow y] \rightarrow x[R_2 \twoheadrightarrow y]$
$\texttt{EquivalentProperties(}R_1\ \texttt{...}\ R_n\texttt{)}$	$\bigwedge_{i \neq j} x[R_j \twoheadrightarrow y] \rightarrow x[R_i \twoheadrightarrow y]$
Mapping OWL DL$^-$ individual assertions to F-Logic	
$\texttt{Individual(}o\ \texttt{type(}C_1\texttt{)}\ \texttt{...}\ \texttt{type(}C_n\texttt{)}$	$\bigwedge o : C_i$
$\texttt{value(}R_1\ o_1\texttt{)}\ \texttt{...}\ \texttt{value(}R_n\ o_n\texttt{))}$	$\bigwedge o[R_i \twoheadrightarrow o_i]$

- Rules with $\bigvee X = o_i$ in the rule body are transformed analogously to Definition 2.9.
- X is a meta-variable and is substituted by the actual variable during the translation

Table 6.1: Axiomatizing OWL DL$^-$ abstract syntax in F-Logic

and a variable as the third argument. This variable is used for relating classes through properties from the `allValuesFrom` and `someValuesFrom` restrictions. Whenever we pass such a value restriction during the translation, a new variable has to be introduced, i.e. x_{new} stands for a freshly introduced variable in every translation step:

Similarly to the transformation presented in Table 2.1, the translation function tr generates a set of Horn clauses where all variables are implicitly universally quantified (after applying similar post-processing steps as defined in Section 2.3.1)[1] Unlike the FOL translation of [Borgida, 1996], we again cannot simply re-use the variable names since nesting of quantifiers is not allowed for Horn clauses.

We split the translation of class axioms into an "inner" part (tr_{lhs}) translating recursively the left- hand side of a subclass relationship to an F-Logic expression, and an "outer" part, tr, translating the whole statement into a (set of) F-Logic Horn clause(s), i.e. Datalog rules. We chose this inner and outer recursion, since parts of the rhs can shift to the rule body during the translation when processing `allValuesFrom` restrictions. A further difference to the OWL DL to FOL mapping presented in the previous section is that we benefit from the semantics of F-Logic in terms of conciseness, since for instance we do not have to axiomatize inheritance or transitivity of the subclass relationship separately.

For object property axioms the translation is easier since after intersections have been processed recursively, the translation is straightforward.

Example 6.1. We will now illustrate the translation by means of a simple OWL DL$^-$ ontology consisting of two axioms:

$$\texttt{Class(A partial B restriction(R allValuesFrom(C)))} \quad (6.1)$$
$$\texttt{ObjectProperty(R super(S) domain(C) range(D) Transitive)} \quad (6.2)$$

We first transform the class description axiom (6.1) to F-Logic by repeatedly applying the transformations in Table 6.1 until no more transformations are possible. We first apply the partial class definition translation (row 1), yielding:

$$tr(x_1 : A, \texttt{intersectionOf(B restriction(R allValuesFrom(C)))}, x_1)$$

Applying the first possible rule in Table 6.1 results in:

$$tr(x_1 : A, \texttt{B}, x_1) \wedge tr(x_1 : A, \texttt{restriction(R allValuesFrom(C))}, x_1)$$

The first part of this conjunction is translated to

$$A :: B$$

and the second part results in

$$tr(x_1 : A \wedge x_1[R \twoheadrightarrow x_2], C, x_2)$$

Now, we can apply the final transformation to

$$A \wedge x_1[R \twoheadrightarrow x_2] \rightarrow x_2 : C$$

and we are done. Next, we translate the property axiom (6.2). From application of the translations in Table 6.1 we obtain the following F-Logic axioms:

[1]Strictly speaking this is not true for the use of `unionOf(`$C_1 \ldots C_n$`)` and `oneOf(`$o_1 \ldots o_n$`)` on the lhs, but disjunctions on the lhs of a rule can be easily rewritten to a set of Horn clauses, using the transformation from [Lloyd and Topor, 1984], cf. also Definition 2.9, and the respective remarks in Section 2.3.1.

$$x[R \twoheadrightarrow y] \rightarrow x[S \twoheadrightarrow y]$$
$$x[R \twoheadrightarrow y] \rightarrow x : C$$
$$x[R \twoheadrightarrow y] \rightarrow y : D$$
$$x[R \twoheadrightarrow y] \wedge y[R \twoheadrightarrow z] \rightarrow x[R \twoheadrightarrow z]$$

□

We conjecture that the semantics of the OWL DL fragment is not changed by our mapping. Our argument is as follows: The original DL semantics and the semantics of our F-Logic translation coincide, since each model of the translation to Datalog in Table 2.1 corresponds to a model of the translation from Table 6.1 and vice versa (because of the separation of vocabulary in OWL DL); the correspondence of the Datalog semantics with the DL semantics on the OWL DL$^-$ fragment has been shown in [Volz, 2004] already.

Note that our mapping for the OWL DL$^-$ variant in the mapping clearly remains in the Horn fragment of F-Logic, and even more precisely in the Datalog fragment, i.e. the FOL semantics and the logic programming semantics of F-Logic mentioned in 2.4 coincide with respect to entailment of ground facts. Note that we could further use a sorted F-Logic, which can separate classes from individuals and properties, but given syntactically correct OWL specifications we do not need this currently, since the translation is only used in one direction and the OWL DL syntax guarantees that the vocabularies are separated.

6.4 OWL Full$^-$ Abstract Syntax

OWL Full adds a number of features on top of OWL DL and also removes some restrictions:

- The vocabulary no longer needs to be separated. This means an identifier can denote a class, an individual and/or a property at the same time. Also, keywords of the language can be used in place of classes, properties and individuals.
- The use of RDF syntax is no longer restricted. OWL DL poses some restrictions on the occurrence of RDF triples. In fact, an RDF document is only valid in OWL DL if it can be translated from OWL abstract syntax according to Section 4.1 in [Patel-Schneider et al., 2004]. On the contrary, any collection of RDF triples is allowed in OWL Full.

Because the use of the vocabulary and the use of triples in OWL Full is not restricted, no computational guarantees can be given. In other words, the entailment problem for OWL Full is in general undecidable, as was shown in [Horrocks et al., 2002] and [Hustadt et al., 2004, Chapter 9]. The former shows that unrestricted use of triples leads to undecidability and the latter shows that the use of language keywords in ontology specification leads to undecidability.

A benefit of leaving out these restrictions is that every valid RDF document is also a valid OWL Full document and every entailment which holds between two RDF graphs also holds for the same RDF graphs under OWL Full semantics. This means there is a complete syntactic and semantic layering of languages between RDFS and OWL Full. On the other hand, there is no complete semantics layering between OWL DL and OWL Full. An RDF graph conforming to the OWL DL syntactical restrictions potentially has more consequences under OWL Full semantics than it has under OWL DL semantics [Patel-Schneider et al., 2004].

Although there is some benefit of allowing arbitrary RDF to be written in an OWL ontology, RDF is often only seen as an exchange syntax for OWL. From that perspective, it should not be a problem if arbitrary RDF is not allowed, as long as both parties are OWL processors. However, if one wants to allow exchange between someone who understands OWL and someone who only understands RDF, it might not be acceptable to live with these limitations.

As opposed to OWL Full we define the OWL Full$^-$ abstract syntax on top of the OWL DL$^-$ abstract syntax. Recall that there is no abstract syntax for the original OWL Full, because it allows arbitrary use of RDF triples, which cannot be captured in the OWL abstract syntax. OWL Full$^-$ imposes some restrictions on the use of RDF triples in order to be able to extend the OWL DL$^-$ abstract syntax.

The only restriction which we relax a little for OWL Full$^-$ with respect to OWL DL$^-$ is the separation of the vocabulary with respect to classes, properties and individuals. In OWL Full$^-$ the vocabularies for classes, properties and instances are no longer separated. Thus, the abstract syntax of OWL Full$^-$ is similar to the abstract syntax of OWL DL$^-$, the only difference being the fact that any identifier can be a class, property and/or individual in OWL Full$^-$.

With this cautious extension we benefit from the ability of treating classes as instances. For example, we can directly use classes as property values, which is a requirement from the Semantic Web Best Practices Working Group[2] [Noy, 2004]. Also, treating classes as instances has been identified as a requirement for ontology mapping. We plan to further elaborate on these issues in WSML Deliverable D20.3.

6.5 OWL Full$^-$ Semantics

With our proposed translation and loosening the restrictions on the vocabulary, we clearly leave the semantics of OWL DL which is based on the $\mathcal{SHOIN}(D)$ Description Logics. However, it can be argued that: Whenever we stay syntactically within OWL DL$^-$ the original DL semantics and the semantics of our F-Logic translation coincide, since the models of the translation from Table 6.1 and the models of the translation to Datalog coincide; correspondence of the Datalog semantics with the DL semantics on the OWL DL$^-$ fragment has been shown in [Volz, 2004] already. On the other hand, by the translation to F-Logic we also implicitly define a clear and easy-to-use semantics for the extension of OWL$^-$ which eliminates the distinction among class-ids, instance-ids and property-ids, since F-Logic does not make this distinction in its vocabulary.

6.6 Limitations of OWL Full$^-$

The major limitations we see currently in OWL Full$^-$ are:

Lack of a notion of value constraints OWL Full$^-$ still lacks a notion of constraints. The universal value restrictions in a class definition do not constrain the values of property fillers.

Lack of cardinality constraints OWL Full$^-$ does not have cardinality constraints. It is not possible to say, for example, that a property must have a

[2] `http://www.w3.org/2001/sw/BestPractices/`

filler for a particular instance, or that there may only be one propertyfiller for a certain instance, e.g. for the name of a person.

Lack of negation OWL Full^{-} does not offer the possibility of deriving negative information and does not allow derivation of inconsistencies. This is because we are taking the approach of a very cautious extension of OWL DL^{-} within the Horn Fragment of F-Logic.

Lack of datatype support OWL Full^{-} does not offer support for datatypes, although we believe that datatypes are required for practical applications.

Unclear interface with RDFS Because OWL Full^{-} is a straightforward extension of OWL DL^{-}, it inherits the translation to RDFS from OWL DL^{-}. However, not all of RDFS is covered by this translation. It is currently unclear how RDFS ontologies which fall outside OWL Full^{-} interact with OWL Full^{-}. The description of such an interface will be part of future work in deliverable D20.3.

We believe that the OWL Full^{-} subset of OWL can serve as a starting point for a more usable Web Ontology language. Extensions of OWL Full^{-} include the addition of Datatypes and Database-like constraints which are discussed in more detail in Deliverable 20.3 of WSMO [de Bruijn et al., 2004]. Furthermore OWL-Full^{-} allows for a straight-forward combination with F-Logic rules.

7 Conclusions and Future Work

In this document we have presented the ontology language OWL Lite$^-$, a variant of the OWL Lite species of the Web Ontology Language OWL. Based on the $\mathcal{L}_0$ language identified by Raphael Volz [Volz, 2004], which is the maximal subset of Description Logics and Datalog, we have defined a proper subset of OWL Lite which allows for efficient reasoning over instances and for easy extension in both the Description Logics (DL) direction and the Logic Programming (LP) direction. As an example, extensions with rules might be necessary for describing the behaviour of Semantic Web Services [Fensel and Bussler, 2002], as the description of the relationships between inputs and outputs requires chaining variables over predicates.

We have defined an extension of OWL Lite$^-$ which allows nominals in value restrictions and introduced syntactical constructs to make hidden features of OWL Lite$^-$ (e.g. intersection of concepts) explicit. This extension, called OWL DL$^-$, is a proper subset of OWL DL and still falls into the intersection of Description Logics and Datalog as defined by the Description Logic $\mathcal{DHL}$. Therefore, OWL DL$^-$ also benefits from efficient query answering and straightforward extensions in both the Descriptions Logics and Logic Programming worlds.

Finally, we have lifted the restriction of the separation between classes and instances in OWL DL$^-$, leading to the OWL Full$^-$ species of OWL$^-$. OWL Full$^-$ is a subset of OWL Full defined as an extension of OWL DL$^-$. As we have seen, although this lifting brings us outside the Description Logics world, an axiomatization in the Datalog fragment of F-Logic still allows us to use efficient query answering engines such as FLORA-2 for reasoning in this fragment of OWL.

The three species of OWL$^-$ described in this document offer an alternative to the OWL species, offering some important features: a) efficient reasoning over instances, b) direct translation into Description Logics and Logic Programming, c) several non-intuitive features of OWL e.g. derivation of equality are removed, d) extensions in both the DL and LP direction are straightforward, and e) the Full species is properly layered.

7.1 Possible Future Extensions of OWL$^-$

We have identified a number of features as possible extensions of OWL$^-$ to be further investigated in the context of WSML deliverable D20.3 OWL Flight[1]:

Unique Name Assumption We do not believe that the unique name assumption holds, as such, on the web. For instance like in a Unix file system, different URIs can denote the same resource or file. Therefore, means for resolving this aspect of the web are essential. However, we do not recommend naively trying to cover this by a counter-intuitive 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 since full equality reasoning by adopting full first-order inferencing comes at high cost. Assuming an external oracle that normalizes term structures then using unique name assumptions could be a more reliable way to go. On the one hand, full support

[1]http://www.wsmo.org/2004/d20/d20.3/

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]). Note that, since OWL$^-$ does not introduce equality, the application of UNA is immaterial in our context. However, and for the reasons previously discussed, future extensions of OWL$^-$ will adhere to the UNA.

(Local) Closed World Reasoning OWL adheres to the Open World Assumption (OWA). 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 absence of possibilities to express CWA will most likely restrict the usefulness of the ontology language for many applications. One way to get 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 in Description Logics [Donini et al., 1998a], which indicates another way to bridge the gap between the logic programming and the description logic worlds.

Constraints As discussed in previous sections, OWL lacks a notion of constraints: value restrictions are used to infer new information about individuals, rather than to detect inconsistencies, and cardinality restrictions are used to derive equality. The introduction of value and cardinality constraints in OWL$^-$ used for detecting inconsistencies instead of deriving new information will be considered.

Datatype support We believe that an appropriate datatype support is required in practical applications. For that reason, an extension of OWL$^-$ with support for datatypes, based on the datatype group approach of OWL-E [Pan and Horrocks, 2004] is a possible future extension. However, this extension does require going beyond plain Datalog, because in Datalog engines one is not allowed to put datatype predicates in the head of a rule. It is possible to classically negate the datatype predicate, thereby moving it to the body of the rule, and interpreting the rule (without a head) as an integrity constraint. However, we have considered in this deliverable only a translation to plain Datalog, which does not have support for integrity constraints.

Some of the above-mentioned features imply going beyond the Description Logic-style semantics of OWL DL and also beyond standard Datalog. Furthermore, *closed world reasoning* and certain types of constraints add 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 begun to explore in the context of OWL$^-$. We plan to take OWL$^-$ as a basis for further extensions towards a more useful ontology language for the Semantic Web.

Acknowledgements

We would especially like to thank the reviewers of the deliverable, Michael Kifer and 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 WSML working group for their advice and input into this document.

Changelog

The following major updates have been done since the December 10th version of the deliverable:

- Review comments by Michael Kifer have been incorporated and the whole document has been proof-read.

Bibliography

[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.

[Balaban, 1995] Balaban, M. (1995). The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15(1):19–60.

[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.

[de Bruijn et al., 2004] de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004). OWL Flight. Deliverable d20.3v0.1, WSML.
Available from http://www.wsmo.org/2004/d20/d20.3/v0.1/.

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

[Decker et al., 1999] Decker, S., Erdmann, M., Fensel, D., and Studer, R. (1999). *Ontobroker: Ontology based Access to Distributed and Semi-Structured Information*. Kluwer Academic.

[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.

[Eiter et al., 2004] Eiter, T., Lukasiewicz, T., Schindlauer, R., and Tompits, H. (2004). Combining answer set programming with description logics for the semantic web. In *Proc. of the International Conference of Knowledge Representation and Reasoning (KR04)*.

[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.

[Gelder et al., 1988] Gelder, A. V., Ross, K., and Schlipf, J. S. (1988). Unfounded sets and well-founded semantics for general logic programs. In *Proceedings 7th ACM Symposium on Principles of Database Systems*, pages 221–230, Austin, Texas. ACM.

[Gelfond and Lifschitz, 1988] Gelfond, M. and Lifschitz, V. (1988). The stable model semantics for logic programming. In Kowalski, R. A. and Bowen, K., editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts. The MIT Press.

[Gelfond and Lifschitz, 1991] Gelfond, M. and Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3/4):365–386.

[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/.

[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, 2003a] Horrocks, I. and Patel-Schneider, P. F. (2003a). 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, 2003b] Horrocks, I. and Patel-Schneider, P. F. (2003b). Three theses of representation in the semantic web. In *Proceedings of the Twelfth International Conference on World Wide Web (WWW2003)*, pages 29–47, Budapest, Hungar.

[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., 2002] Horrocks, I., Patel-Schneider, P. F., and van Harmelen, F. (2002). Reviewing the design of DAML+OIL: An ontology language for the semantic web. In *Proc. of the 18th Nat. Conf. on Artificial Intelligence (AAAI 2002).*

[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.

[Khoshafian and Copeland, 1986] Khoshafian, S. and Copeland, G. (1986). Object identity. In *Proc. of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'86)*, pages 406–416.

[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). Query answering for OWL-DL with rules. In *Proceedings of 3rd International Semantic Web Conference (ISWC2004)*, Hiroshima, Japan.

[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/)*.

[Noy, 2004] Noy, N. (2004). Representing classes as property values on the semantic web. Working Draft 29 July 2004, W3C. Available from http://www.w3.org/TR/swbp-classes-as-values/.

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

[Pan and Horrocks, 2003b] Pan, J. Z. and Horrocks, I. (2003b). RDFS(FA) and RDF MT: Two semantics for RDFS. In *Proceedings of the 2nd International Semantic Web Conference (ISWC2003)*, pages 30–46.

[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.

[Przymusinski, 1986] Przymusinski, T. C. (1986). On the semantics of stratified deductive databases. In *Proceedings of the Workshop on the Foundations of Deductive Databases and Logic Programming*, pages 433–443, Washington, DC, USA.

[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.

[Yang et al., 2003a] Yang, G., Kifer, M., and Zhao, C. (2003a). FLORA-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Proceedings of the Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, Catania, Sicily, Italy.

[Yang et al., 2003b] Yang, G., Kifer, M., and Zhao, C. (2003b). FLORA-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *Proceedings of the Second International Conference on Ontologies, Databases and Applications of Semantics (ODBASE)*, Catania, Sicily, Italy.

A Appendix: OWL$^-$ Abstract Syntax

We give here the Abstract syntax of OWL DL$^-$ and OWL Full$^-$ (a restriction of the OWL DL abstract syntax) in the style of [Patel-Schneider et al., 2004]. The only difference between OWL DL$^-$ and OWL Full$^-$ is that in OWL Full$^-$ there is no distinction between class, individual and property identifiers.

In the following, a symbol between square brackets ([]) is not required to occur (may occur 0 or 1 time). A symbol between curly brackets ({}) may occur zero or more times. A symbol not enclosed in square or curly brackets is required and may only occur one time. The bar (|) stands for an exclusive choice. All keywords are represented in boldface.

An ontology in OWL DL$^-$ consists of a number of so-called directives, which can either be annotations, axioms or facts.

ontology ::= 'Ontology(' [**ontologyID**] { **directive** } ')'
directive ::= 'Annotation(' **ontologyPropertyID ontologyID** ')'
| 'Annotation(' **annotationPropertyID URIreference** ')'
| 'Annotation(' **annotationPropertyID dataLiteral** ')'
| 'Annotation(' **annotationPropertyID individual** ')'
| **axiom**
| **fact**

These are the different types of identifiers in OWL DL$^-$.

individualID ::= **URIreference**
classID ::= **URIreference**
ontologyID ::= **URIreference**
individualvaluedPropertyID ::= **URIreference**
annotationPropertyID ::= **URIreference**
ontologyPropertyID ::= **URIreference**

The non-functional properties of all the elements in the ontology are described using so-called annotations. All axioms and facts in the ontology have the possibility of attaching annotations.

annotation ::= 'annotation(' **annotationPropertyID URIreference** ')'
| 'annotation(' **annotationPropertyID dataLiteral** ')'
| 'annotation(' **annotationPropertyID individual** ')'

Facts consist of information about individuals. Note that we do not allow the assertion of individual (in)equality, because we do not have equality in the underlying formalism and an OWL DL$^-$ ontology adheres to the unique name assumption.

fact ::= **individual**
individual ::= 'Individual(' [**individualID**] { **annotation** } { 'type(' **type** ')' }
{ **value** } ')'

value ::= 'value(' **individualvaluedPropertyID individualID** ')'
| 'value(' **individualvaluedPropertyID individual** ')'
type ::= **description**

These are the literals we allow (note that literals are only allowed in annotation properties, because OWL DL$^-$ does not have datatypes).

dataLiteral ::= **typedLiteral** | **plainLiteral**
typedLiteral ::= **lexicalForm^^URIreference**
plainLiteral ::= **lexicalForm** | **lexicalForm@languageTag**
lexicalForm ::= as in RDF, a unicode string in normal form C
languageTag ::= as in RDF, an XML language tag

We allow different types of class axioms. Our class axioms diverge somewhat from the OWL AS class axioms, because we distinguish between partial/complete class descriptions and because we distinguish between descriptions allowed on the left/right-hand side of the GCI (General Class Inclusion, `SubClassOf`).

Note also that we do not allow enumerated classes.

axiom ::= 'Class(' **classID** ['Deprecated'] 'partial' { **annotation** }
{ **rhs_description** } ')'
axiom ::= 'Class(' **classID** ['Deprecated'] 'complete' { **annotation** }
{ **description** } ')'
axiom ::= | 'EquivalentClasses(' **description** { **description** } ')'
| 'SubClassOf(' **lhs_description rhs_description** ')'

A description is allowed to occur both on the left- and the right-hand side of the GCI. A lhs_description can only occur on the left-hand side; a rhs_description can only occur on the right-hand side.

Note that for cardinality, we only allow a minimal cardinality of one on the left-hand side.

description ::= **classID**
| **restriction**
| 'intersectionOf(' { **description** } ')'
lhs_description ::= **description**
| **lhs_restriction**
| 'unionOf(' { **lhs_description** } ')'
| 'oneOf(' { **individualID** } ')'
rhs_description ::= **description**
| **rhs_restriction**
restriction ::= 'restriction(' **individualvaluedPropertyID**
individualRestrictionComponent
{ **individualRestrictionComponent** } ')'
individualRestrictionComponent ::= 'value(' **individualID** ')'
lhs_restriction ::= **restriction**
| 'restriction(' **individualvaluedPropertyID**
lhs_individualRestrictionComponent
{ **lhs_individualRestrictionComponent** } ')'
lhs_individualRestrictionComponent ::= 'someValuesFrom(' **lhs_description** ')'
| 'minCardinality(1)'
rhs_restriction ::= **restriction**

```
                          | 'restriction(' individualvaluedPropertyID
                                  rhs_individualRestrictionComponent
                                  { rhs_individualRestrictionComponent } ')'
rhs_individualRestrictionComponent ::= 'allValuesFrom(' rhs_description ')'


axiom ::= 'ObjectProperty(' individualvaluedPropertyID ['Deprecated']
                    { annotation }
                    { 'super(' individualvaluedPropertyID ')' }
                    [ 'inverseOf(' individualvaluedPropertyID ')' ]
                    [ 'Symmetric' ] [ 'Transitive' ]
                    { 'domain(' description ')' } { 'range(' description ')' } ')'
            | 'AnnotationProperty(' annotationPropertyID
                    { annotation } ')'
            | 'OntologyProperty(' ontologyPropertyID { annotation } ')'
axiom ::= 'EquivalentProperties(' individualvaluedPropertyID
                    individualvaluedPropertyID
                    { individualvaluedPropertyID } ')'
            | 'SubPropertyOf(' individualvaluedPropertyID
                    individualvaluedPropertyID ')'
```