W3C

Web Ontology Language (OWL)
Abstract Syntax and Semantics

Editor's Draft 16 January 2003

This version:
http://www.w3.org/TR/2002/WD-owl-semantics-????????/
Latest version:
http://www.w3.org/TR/owl-semantics/
Previous version:
http://www.w3.org/TR/2002/WD-owl-semantics-20021108/
Editors:
Peter F. Patel-Schneider, Bell Labs Research, Lucent Technologies
Patrick Hayes, IHMC, University of West Florida
Ian Horrocks, Department of Computer Science, University of Manchester

The normative form of this document is a compound HTML document.

Copyright © 2003 W3C® (MIT, ERCIM, Keio), All Rights Reserved. W3C liability, trademark, document use and software licensing rules apply.

Abstract

This description of OWL, the Web Ontology Language being designed by the W3C Web Ontology Working Group, contains a high-level abstract syntax for both OWL DL and OWL Lite, sublanguages of OWL. A model-theoretic semantics is given to provide a formal meaning for OWL ontologies written in this abstract syntax. A model-theoretic semantics in the form of an extension to the RDFS model theory is also given to provide a formal meaning for OWL ontologies as RDF graphs. A mapping from the abstract syntax to RDF graphs is given and the two model theories are shown to have the same consequences on OWL ontologies that can be written in the abstract syntax.

Status of this document

This is [[an editor's revision of]] a W3C Web Ontology Working Group Working Draft produced 8 November 2002 as part of the W3C Semantic Web Activity (Activity Statement). It incorporates decisions made by the Working Group in designing the OWL Web Ontology Language. This is [[an editor's revision of]] a public W3C Working Draft and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite as other than "work in progress". A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR/.

There are no patent disclosures related to this work at the time of this writing.

Comments on this document are invited and should be sent to the public mailing list public-webont-comments@w3.org. An archive of comments is available at http://lists.w3.org/Archives/Public/public-webont-comments/.

Table of contents

- 1. Introduction
  - 1.1 Differences from DAML+OIL
- 2. Abstract Syntax
  - 2.1 Ontologies
  - 2.2 Facts
  - 2.3 Axioms
    - 2.3.1 OWL Lite Axioms
      - 2.3.1.1 OWL Lite Class Axioms
      - 2.3.1.2 OWL Lite Restrictions
      - 2.3.1.3 OWL Lite Property Axioms
    - 2.3.2 OWL DL Axioms
      - 2.3.2.1 OWL DL Class Axioms
      - 2.3.2.2 OWL DL Descriptions
      - 2.3.2.3 OWL DL Restrictions
      - 2.3.2.4 OWL DL Property Axioms
- 3. Direct Model-Theoretic Semantics
  - 3.1 Vocabularies and Interpretations
  - 3.2 Interpreting Embedded Constructs
  - 3.3 Interpreting Axioms and Facts
  - 3.4 Interpreting Ontologies
- 4. Mapping to RDF Graphs
  - 4.1 Translation to RDF Graphs
  - 4.2 Definition of OWL DL and OWL Lite Ontologies in RDF Graph Form
- 5. RDFS-Compatible Model-Theoretic Semantics
  - 5.1 The OWL and RDFS universes
    - 5.1.1 Qualified Names and URI References RDFS universes
  - 5.2 OWL Interpretations
  - 5.3 OWL DL
    - 5.3.1 OWL DL Entailment
    - 5.3.2 Correspondence to the Direct Semantics
  - 5.4 OWL Full

- Appendix A. Proofs (Informative)
  - A.1 Correspondence between Abstract Syntax and OWL DL
    - A.1.1 Lemma 1

        - A.1.2 Lemma 2
        - A.1.3 Lemma 3
        - A.1.4 Lemma 4
        - A.1.5 Correspondence Theorem
    - A.2 Correspondence between OWL DL and OWL Full
- Appendix B. Examples (Informative)
    - B.1 Examples of Mapping from Abstract Syntax to RDF Graphs
    - B.2 Examples of Entailments in OWL DL and OWL Full
- Index of Vocabulary (Informative)
- References
    - Normative References
    - Other References

---

1. Introduction

This document contains several interrelated specifications of the several styles of OWL, the Web Ontology Language being produced by the W3C Web Ontology Working Group (WebOnt). First, Section 2 contains a high-level, abstract syntax for both OWL Lite, a subset of OWL and a fuller style of using OWL, but one that still places some limitations on how OWL ontologies are constructed, called OWL DL. Eliminating these limitations results in the full OWL language, called OWL Full, which has the same syntax as RDF. The normative exchange syntax for OWL is RDF/XML [*RDF Syntax*]; the OWL Reference document [*OWL Reference*] shows how the RDF syntax is used in OWL. A mapping from the OWL abstract syntax to RDF graphs [*RDF Concepts*] is, however, provided in Section 4.

This document contains two formal semantics for OWL. One of these semantics, defined in Section 3, is a direct, standard model-theoretic semantics for OWL ontologies written in the abstract syntax. The other, defined in Section 5, is a vocabulary extension of the RDF model-theoretic semantics [*RDF MT*] that provides semantics for OWL ontologies in the form of RDF graphs. Two versions of this second semantics are provided, one that corresponds more closely to the direct semantics (and is thus a semantics for OWL DL) and one that can be used in cases where classes need to be treated as individuals or other situations that cannot be handled in the abstract syntax (and is thus a semantics for OWL Full). These two versions are actually very close, only differing in how they divide up the domain of discourse.

Appendix A contains a proof that the direct and RDFS-compatible semantics have the same consequences on OWL ontologies that correspond to abstract OWL ontologies that separate OWL individuals, OWL classes, OWL properties, and the RDF, RDFS, and OWL structural vocabulary. For such OWL ontologies the direct model theory is authoritative and the RDFS-compatible model theory is secondary. Finally a few examples of the various concepts defined in the document are presented in Appendix B.

This document is designed to be read by those interested in the technical details of OWL. It is not particularly intended for the casual reader. Developers of parsers and other syntactic tools for OWL will be particularly interested in Sections 2 and 4. Developers of reasoners and other semantic tools for OWL will be particulary interested in Sections 3 and 5.

1.1. Differences from DAML+OIL

The language described in this document is very close to the DAML+OIL web ontology language [*DAML+OIL*]. The only substantive changes between OWL and DAML+OIL are

- the removal of qualified number restrictions;
- the ability to directly state that properties can be symmetric;
- a new construct for stating that several individuals are distinct; and
- the absence in the abstract syntax of some abnormal DAML+OIL constructs, particularly restrictions with extra components.

There are also a number of minor differences between OWL and DAML+OIL, including a number of changes to the names of the various constructs, as mentioned in Appendix A of the OWL Reference Description [*OWL Reference*].

Acknowledgments

The Joint US/EU ad hoc Agent Markup Language Committee developed DAML+OIL, which is the direct precursor to OWL. Many of the ideas in DAM+OIL and thus in OWL are also present in the Ontology Inference Layer (OIL). Many of the other members of the W3C Web Ontology Working Group have had substantial input into this document.

Index of Vocabulary (Informative)

The following table provides pointers to information about each element of the OWL vocabulary. The first column points to the vocabulary element's major definition in the abstract syntax of Section 2. The second column points to the vocabulary element's major definition in the OWL Lite abstract syntax. The third column points to the vocabularly element's major definition in the direct semantics of Section 3. The fourth column points to the major piece of the translation from the abstract syntax to triples for the vocabulary element Section 4. The fifth column points to the vocabularly element's major definition in the RDFS-compatible semantics of Section 5.

Vocabulary Terms

Vocabulary Term	Abstract OWL DL Syntax	Abstract OWL Lite Syntax	Direct Semantics	Mapping to Triples	RDFS-Compatible Semantics
owl:AllDifferent				4.1	5.2
owl:allValuesFrom	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:backwardCompatibleWith	2.1	2.1		4.1	

Vocabulary Term	Abstract OWL DL Syntax	Abstract OWL Lite Syntax	Direct Semantics	Mapping to Triples	RDFS-Compatible Semantics
owl:cardinality	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:Class	2.3.2.1	2.3.1.1	3.3	4.1	5.2
owl:complementOf	2.3.2.2		3.2	4.1	5.2
owl:DatatypeProperty	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:DeprecatedClass					
owl:DeprecatedProperty					
owl:differentFrom	2.2	2.2	3.3	4.1	5.2
owl:disjointWith	2.3.2.1		3.3	4.1	5.2
owl:distinctMembers				4.1	5.2
owl:FunctionalProperty	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:hasValue	2.3.2.3		3.2	4.1	5.2
owl:imports	2.1	2.1	3.4	4.1	5.3.1
owl:incompatibleWith	2.1	2.1		4.1	
owl:intersectionOf	2.3.2.2		3.2	4.1	5.2
owl:InverseFunctionalProperty	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:inverseOf	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:maxCardinality	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:minCardinality	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:Nothing	2.1		3.2	4.1	5.2
owl:ObjectProperty	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:oneOf	2.3.2.2		3.2	4.1	5.2
owl:onProperty	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:Ontology	2.1	2.1	3.4	4.1	5.2
owl:priorVersion	2.1	2.1		4.1	
owl:Property					5.2

Vocabulary Term	Abstract OWL DL Syntax	Abstract OWL Lite Syntax	Direct Semantics	Mapping to Triples	RDFS-Compatible Semantics
owl:Restriction	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:sameAs					5.2
owl:sameClassAs	2.3.2.1	2.3.1.1	3.3	4.1	5.2
owl:sameIndividualAs	2.2	2.2	3.3	4.1	5.2
owl:samePropertyAs	2.3.1.3	2.3.1.3	3.3	4.1	5.2
owl:someValuesFrom	2.3.2.3	2.3.1.2	3.2	4.1	5.2
owl:SymmetricProperty	2.3.2.4	2.3.1.3	3.3	4.1	4.2
owl:Thing	2.1	2.1	3.2	4.1	5.2
owl:TransitiveProperty	2.3.2.4	2.3.1.3	3.3	4.1	5.2
owl:unionOf	2.3.2.2		3.2	4.1	5.2
rdf:List					5.2
rdf:nil					5.2
rdf:type	2.2	2.2	3.3	4.1	
rdfs:comment	2.1	2.1		4.1	
rdfs:Datatype					5.2
rdfs:domain	2.3.2.4	2.3.1.3	3.3	4.1	5.2
rdfs:label	2.1	2.1		4.1	
rdfs:Literal					5.2
rdfs:range	2.3.2.4	2.3.1.3	3.3	4.1	5.2
rdfs:subClassOf	2.3.2.1	2.3.1.1	3.3	4.1	
rdfs:subPropertyOf	2.3.1.3	2.3.1.3	3.3	4.1	

References

Normative References

[RDF Concepts]
Resource Description Framework (RDF): Concepts and Abstract Syntax. Graham Klyne and Jeremy J. Carroll, eds. W3C Working Draft 08 November 2002. Latest version is available at http://www.w3.org/TR/rdf-concepts/.

[RDF MT]
RDF Semantics. Patrick Hayes, ed. W3C Working Draft 12 November 2002. Latest version is available at http://www.w3.org/TR/rdf-mt/.

[RDF Syntax]
RDF/XML Syntax Specification (Revised) Dave Beckett, ed. W3C Working Draft 8 November 2002. Latest version is available at http://www.w3.org/TR/rdf-syntax-grammar/.

[XML]
Extensible Markup Language (XML) 1.0 (Second Edition). Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler, eds. W3C Recommendation 6 October 2000. Latest version is available at http://www.w3.org/TR/REC-xml/.

[XML Schema Datatypes]
XML Schema Part 2: Datatypes.. Paul V. Biron and Ashok Malhotra, eds. W3C Recommendation 02 May 2000. Latest version is available at http://www.w3.org/TR/REC-xmlschema-2/.

Other References

[DAML+OIL]
DAML+OIL (March 2001) Reference Description. Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. W3C Note 18 December 2001. Latest version is available at http://www.w3.org/TR/daml+oil-reference.

[OWL Features]
Feature Synopsis for OWL Lite and OWL. Deborah L. McGuinness and Frank van Harmelen. W3C Working Draft 29 July 2002. Latest version is available at http://www.w3.org/TR/owl-features/.

[OWL Issues]
Web Ontology Issue Status. Michael K. Smith, ed. 10 July 2002.

[OWL Reference]
OWL Web Ontology Language 1.0 Reference. Mike Dean, Dan Connolly, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. W3C Working Draft 29 July 2002. Latest version is available at http://www.w3.org/TR/owl-ref/.

[RDFMS]
Resource Description Framework (RDF) Model and Syntax Specification. Ora Lassila and Ralph R. Swick, eds. W3C Recommendation 22 February 1999. Latest version is available at http://www.w3.org/TR/REC-rdf-syntax/.

[RDF Schema]
RDF Vocabulary Description Language 1.0: RDF Schema. Dan Brickley and R. V. Guha, eds. W3C Working Draft 12 November 2002. Latest version is available at http://www.w3.org/TR/rdf-schema/.

[RDF Tests]
RDF Test Cases. Jan Grant and Dave Beckett, eds. W3C Working Draft 12 November 2002. Latest version is available at http://www.w3.org/TR/rdf-testcases/.

2. Abstract Syntax

The description of OWL in this Section abstracts from concrete syntax and thus facilitates access to and evaluation of the language. A high-level syntax is used to make the language features easier to see. This particular syntax has a frame-like style, where a collection of information about a class or property is given in one large syntactic construct, instead of being divided into a number of atomic chunks (as in most Description Logics) or even being divided into even more triples (as when writing OWL as RDF graphs [*RDF Concepts*]). The syntax used here is rather informal, even for an abstract syntax - in general the arguments of a construct should be considered to be unordered wherever the order would not affect the meaning of the construct.

The abstract syntax is specified here by means of a version of Extended BNF, very similar to the EBNF notation used for XML [*XML*]. Terminals are quoted; non-terminals are not quoted. Alternatives are either separated by vertical bars (|) or are given in different productions. Components that can occur at most once are enclosed in square brackets ([…]); components that can occur any number of times (including zero) are enclosed in braces ({…}). Whitespace is ignored in the productions here.

The meaning of each construct in the abstract syntax is described when it is introduced. The formal meaning of these constructs is given in Section 3 via a model-theoretic semantics.

While it is widely appreciated that all of the features in expressive languages such as OWL are important to some users, it is also understood that such languages may be daunting to some groups who are trying to support a tool suite for the entire language. In order to provide a simpler target for implementation, a smaller language has been defined, called OWL Lite [*OWL Features*]. This smaller language also attempts to describe a useful language that provides more than RDF Schema [*RDF Schema*] with the goal of adding functionality that is important in order to support web applications. The abstract syntax is expressed both for this smaller language, called the OWL Lite abstract syntax here, and also for a fuller style of OWL, called the OWL DL abstract syntax here.

The abstract syntax here is less general than the exchange syntax for OWL. In particular, it does not permit the construction of self-referential syntactic constructs. It is also intended for use in cases where classes, properties, and individuals form disjoint collections. These are roughly the limitations required to make reasoning in OWL be decidable, and thus this abstract syntax should be thought of a syntax for OWL DL.

OWL uses some of the facilities of XML Schema. [*XML Schema Datatypes*]. The following built-in non-list XML Schema datatypes can be used in OWL by means of the XML Schema canonical URI reference for the datatype, http://www.w3.org/2001/XMLSchema#name, where name is the local name of the datatype: xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte and xsd:positiveInteger. The other built-in XML Schema datatypes may be used, but with caveats (see below).

The specific considerations with the other built-in XML Schema datatypes are:

xsd:duration,
: In the current version of XML Schema no equality function is defined for xsd:duration. This may give surprising results when combined with OWL cardinality restrictions. Later revisions of XML Schema datatypes are expected to provide such a function, in which case the revised duration datatype would be fully appropriate for use with OWL.

xsd:QName,
xsd:ENTITY,
: These datatypes require an enclosing XML document context, which may not be available in a specific application scenario for an OWL ontology.

xsd:NOTATION,
: This datatype is intended for use as a base type for user defined datatypes.

xsd:ID,
xsd:IDREF,
: ID and IDREF are for cross references within an XML document and thus are not useful in OWL.

xsd:IDREFS,
xsd:ENTITIES,
xsd:NMTOKENS,
: List valued datatypes are not used in OWL.

2.1. Ontologies

An OWL ontology in the abstract syntax is a sequence of axioms and facts, plus inclusion references to other ontologies, which are considered to be included in the ontology. Ontologies can also have annotations that can be used to record authorship and other information associated with an ontology. OWL ontologies are web documents, and can be referenced by means of a URI.

```
ontology ::= 'Ontology(' { directive } ')'

directive ::= 'Annotation(' URIreference URIreference ')'
            | 'Annotation(' URIreference dataLiteral ')'
            | 'Imports(' URI ')'
            | axiom
            | fact
```

Ontologies incorporate information about classes, properties, and individuals, each of which can have an identifier which is a URI reference.

```
datatypeID                 ::= URIreference
classID                    ::= URIreference
individualID               ::= URIreference
datavaluedPropertyID       ::= URIreference
individualvaluedPropertyID ::= URIreference
```

If a URI reference is a datatype, i.e., the URI reference points to one of the useful XML Schema datatypes described above, then that URI reference cannot be used as the identifier of a class. However, a URI reference can be the identifier of a class or datatype as well as the identifier of a property as well as the identifier of an individual in this abstract syntax, although the ontology cannot then be translated into an

OWL DL RDF graph. Individual identifiers are used to refer to resources, and data literals are used to refer to data values.

In OWL, as in RDF, a datatype denotes the set of data values that is the value space for the datatype. Classes denote sets of individuals. Properties relate individuals to other information, and are divided into two disjoint groups, data-valued properties and individual-valued properties. Data-valued properties relate individuals to data values; individual-valued properties relate individuals to other individuals.

There are two built-in classes in OWL, they both use URI references in the OWL URI, `http://www.w3.org/2002/07/owl#`, for which the namespace name `owl` is used here. The class with identifier `owl:Thing` is the class of all individuals, and is part of OWL Lite. The class with identifier `owl:Nothing` is the empty class.

Many OWL constructs use annotations, which, just like annotation directives, are used to record information associated with some portion of the construct.

```
annotation ::= 'annotation(' URIreference URIreference ')'
             | 'annotation(' URIreference dataLiteral ')'
```

2.2. Facts

There are two kinds of facts in the OWL abstract syntax. The first kind of fact states information about a particular individual, in the form of classes that the individual belongs to plus properties and values of that individual. An individual can be given an individualID that will denote that individual, and can be used to refer to that individual. However, an individual need not be given an individualID. Such individuals are anonymous (blank in RDF terms) and cannot be directly referred to elsewhere. The syntax here is set up to mirror RDF/XML syntax without the use of rdf:nodeID.

```
fact ::= individual
individual ::= 'Individual(' [ individualID ] { annotation }
                             { 'type(' type ')' } { propertyValue } ')'
propertyValue ::= 'value(' individualvaluedPropertyID  individualID ')'
                | 'value(' individualvaluedPropertyID  individual ')'
                | 'value(' datavaluedPropertyID  dataLiteral ')'
```

Facts are the same in the OWL Lite and OWL DL abstract syntaxes, except for what can be a type. In OWL Lite, types can be classIDs or OWL Lite restrictions

```
type ::= classID
       | restriction
```

In the OWL DL abstract syntax types can be general descriptions, which include classIDs and OWL Lite restrictions as well as other constructs

```
type ::= description
```

Data literals in the abstract syntax consist of a datatype and the lexical representation of a data value in that datatype (a typed literal) or just a string (an untyped literal). In the abstract syntax typed literals are all valid, i.e., xsd:integer1.5 is not a data literal.

```
dataLiteral ::= typedLiteral | untypedLiteral
typedLiteral ::= datatypeID lexicalForm
untypedLiteral ::= lexicalForm
lexicalForm ::= UniCode string enclosed in quotes
```

In the abstract syntax the other kinds of facts are used to make individual identifiers be the same or pairwise distinct.

```
fact ::= 'SameIndividual(' individualID {individualID} ')'
       | 'DifferentIndividuals(' individualID {individualID} ')'
```

2.3. Axioms

The biggest differences between the OWL Lite and OWL DL abstract syntaxes show up in the axioms, which are used to provide information about classes and properties. As it is the smaller language, OWL Lite axioms are given first, and then OWL DL axioms are given as additions or modifications to OWL Lite.

Axioms are used to associate class and property identifiers with either partial or complete specifications of their characteristics, and to give other logical information about classes and properties. Axioms used to be called definitions, but they are not all definitions in the common sense of the term, as has been made evident in several discussions in the WG, and thus a more-neutral name has been chosen.

The syntax used here is meant to look somewhat like the syntax used in some frame systems. Each class axiom in OWL Lite contains a collection of more-general classes and a collection of local property restrictions in the form of restriction constructs. The restriction construct gives the local range of a property, how many values are permitted, and/or a collection of required values. The class is made either equivalent to or a subset of the intersection of these more-general classes and restrictions. In the OWL DL abstract syntax a class axiom contains a collection of descriptions, which can be more-general classes, restrictions, sets of individuals, and boolean combinations of descriptions. Classes can also be specified by enumeration or be made equivalent or disjoint.

Properties can be equivalent to or sub-properties of others; can be made functional, inverse functional, symmetric, or transitive; and can be given global domains and ranges. However, most information about properties is more naturally expressed in restrictions, which allow local range and cardinality information to be specified.

There is no requirement in the abstract syntax that there be an axiom for each class used in an ontology. In fact, there can be any number of axioms for each class, including none. Properties used in an abstract syntax ontology have to be categorized as either data-valued or individual-valued, so they need an axiom for this purpose at least. There is no requirement that there be at most one axiom for a class or property used in an ontology. Each axiom for a particular class (or property) identifier contributes to the meaning of the class (or property).

2.3.1. OWL Lite Axioms

2.3.1.1. OWL Lite Class Axioms

In OWL Lite class axioms are used to state that a class is exactly equivalent to, for the modality `complete`, or a subclass of, for the modality `partial`, the conjunction of a collection of superclasses and OWL Lite Restrictions.

```
axiom ::= 'Class(' classID modality { annotation } { super } ')'
modality ::= 'complete' | 'partial'
super ::= classID | restriction
```

In OWL Lite it is possible to state that two classes are equivalent.

```
axiom ::= 'EquivalentClasses(' classID { classID } ')'
```

2.3.1.2. OWL Lite Restrictions

Restrictions are used in OWL Lite class axioms to provide local constraints on properties in the class. Each allValuesFrom part of a restriction makes the constraint that all values of the property for individuals in the class must belong to the specified class or datatype. Each someValuesFrom part makes the constraint that there must be at least one value for the property that belongs to the specified class or datatype. The cardinality part says how many distinct values there are for the property for each individual in the class. In OWL Lite the only cardinalities allowed are 0 and 1.

There is a side condition in OWL Lite that properties that are transitive or that have transitive sub-properties may not have cardinality conditions expressed on them in restrictions.

```
restriction ::= 'restriction(' datavaluedPropertyID { 'allValuesFrom(' datatypeID ')'}
                                { 'someValuesFrom(' datatypeID ')' } { cardinality } ')'
              | 'restriction(' individualvaluedPropertyID { 'allValuesFrom(' classID ')'}
                                { 'someValuesFrom(' classID ')' } { cardinality } ')'
cardinality ::= 'minCardinality(0)' | 'minCardinality(1)' |
              | 'maxCardinality(0)' | 'maxCardinality(1)' |
              | 'cardinality(0)'    | 'cardinality(1)'
```

2.3.1.3. OWL Lite Property Axioms

Properties are also specified using a frame-like syntax. Properties are divided into data-valued properties, which relate individuals to data values, like integers, and individual-valued properties, which relate individuals to other individuals. Properties can be given super-properties, allowing the construction of a property hierarchy. It does not make sense to have an individual property be a super-property of a data property, or vice versa.

Properties can also be given domains and ranges. A domain for a property specifies which individuals are potential subjects of statements that have the property as predicate, just as in RDFS. In OWL Lite the domains of properties are classes. Properties can have multiple domains, in which case only individuals that belong to all of the domains are potential subjects. A range for a property specifies which individuals or data values can be objects of statements that have the property as predicate. Again, properties can have multiple ranges, in which case only individuals or data values that belong to all of the ranges are potential

objects. In OWL Lite ranges for individual-valued properties are classes; ranges for data-valued properties are datatypes.

Data-valued properties can be specified as (partial) functional, i.e., given an individual, there can be at most one relationship to a data value for that individual in the property. Individual-valued properties can be specified to be the inverse of another property. Individual-valued properties can also be specified to be symmetric as well as functional, inverse functional, or transitive.

Individual-valued properties that are transitive, or that have transitive sub-properties, may not have cardinality conditions expressed on them, either in restrictions or by being functional, or inverse functional. This is needed to maintain the decidability of the language.

```
axiom ::= 'DatatypeProperty(' datavaluedPropertyID { annotation } { 'super(' datavaluedPropertyID ')' }
                              {domain(classID')'} {range(datatypeID')' }
                              [Functional] ')'

axiom ::= 'ObjectProperty(' individualvaluedPropertyID { annotation } { 'super(' individualvaluedPropertyID ')'}
             { 'domain(' classID ')'} { 'range(' classID ')' }
             ['inverseOf(' individualvaluedPropertyID' )'] [ Symmetric ]
             [ 'Functional' | 'InverseFunctional' | 'Functional' 'InverseFunctional' | 'Transitive' ] ')'
```

The following axioms make several properties be equivalent, or make one property be a sub-property of another.

```
axiom ::= 'EquivalentProperties(' datavaluedPropertyID  { datavaluedPropertyID } ')'
        | 'SubPropertyOf(' datavaluedPropertyID  datavaluedPropertyID ')'
        | 'EquivalentProperties(' individualvaluedPropertyID  { individualvaluedPropertyID } ')'
        | 'SubPropertyOf(' individualvaluedPropertyID  individualvaluedPropertyID ')'
```

2.3.2. OWL DL Axioms

2.3.2.1. OWL DL Class Axioms

The OWL DL abstract syntax has more-general versions of the OWL Lite class axioms where superclasses, more-general restrictions, and boolean combinations of these are allowed. Together, these constructs are called descriptions.

```
axiom ::= 'Class(' classID modality { annotation } { description } ')'
modality ::= 'complete' | 'partial'
```

In the OWL DL abstract syntax it is also possible to make a class exactly consist of a certain set of individuals, as follows.

```
axiom ::= 'EnumeratedClass(' classID { annotation } { individualID } ')'
```

Finally, in the OWL DL abstract syntax it is possible to require that a collection of descriptions be pairwise disjoint, or have the same instances, or that one description is a subclass of another. Note that the last two of these axioms generalize the first kind of class axiom just above.

```
axiom ::= 'DisjointClasses(' description { description } ')'
        | 'EquivalentClasses(' description { description } ')'
        | 'SubClassOf(' description description ')'
```

2.3.2.2. OWL DL Descriptions

Descriptions in the OWL DL abstract syntax include class identifiers and the restriction constructor. Descriptions can also be boolean combinations of other descriptions, and sets of individuals.

```
description ::= classID
              | restriction
              | 'unionOf(' { description } ')'
              | 'intersectionOf(' { description } ')'
              | 'complementOf(' description ')'
              | 'oneOf(' { individualID } ')'
```

2.3.2.3. OWL DL Restrictions

Restrictions in the OWL DL abstract syntax generalize OWL Lite restrictions by allowing descriptions where classes are allowed in OWL Lite and allowing sets of data values as well as datatypes. The combination of datatypes and sets of data values is called a data range. In the OWL DL abstract syntax, values can also be given for properties in classes. As well, cardinalities are not restricted to only 0 and 1.

```
restriction ::= 'restriction(' datavaluedPropertyID { 'allValuesFrom(' dataRange ')'}
                               { 'someValuesFrom(' dataRange ')'} { 'value(' dataLiteral ')' }
                               { cardinality } ')'
              | 'restriction(' individualvaluedPropertyID { 'allValuesFrom(' description ')'}
                               { 'someValuesFrom(' description ')'} { 'value(' individualID ')' }
                               { cardinality } ')'
cardinality ::= 'minCardinality(' non-negative-integer ')'
              | 'maxCardinality(' non-negative-integer ')'
              | 'cardinality(' non-negative-integer ')'
```

A dataRange, used as the range of a data-valued property and in other places in the OWL DL abstract syntax, is either a datatype or a set of data values.

```
dataRange ::= datatypeID
            | 'oneOf(' { dataLiteral } ')'
```

As in OWL Lite, there is a side condition that properties that are transitive, or that have transitive sub-properties, may not have cardinality conditions expressed on them in restrictions.

2.3.2.4. OWL DL Property Axioms

Property axioms in the OWL DL abstract syntax generalize OWL Lite property axioms by allowing descriptions in place of classes and data ranges in place of datatypes in domains and ranges.

```
axiom ::= 'DatatypeProperty(' datavaluedPropertyID { annotation } { 'super(' datavaluedPropertyID ')'}
                              { 'domain(' description ')' } { 'range(' dataRange ')'}
                              [ 'Functional' ] ')'
        | 'ObjectProperty(' individualvaluedPropertyID { annotation } { 'super(' individualvaluedPropertyID ')'}
               { 'domain(' description ')' } { 'range(' description ')' }
               [ 'inverseOf(' individualvaluedPropertyID ')' ] [ 'Symmetric' ]
               [ 'Functional' | 'InverseFunctional' | 'Functional' 'InverseFunctional' | 'Transitive' ] ')'
```

3. Direct Model-Theoretic Semantics

This model-theoretic semantics for OWL goes directly from ontologies in the abstract syntax to a standard model theory. It is simpler than the semantics in Section 5 for RDF graphs that is a vocabulary extension of the RDFS model theory.

3.1. Vocabularies and Interpretations

The semantics here starts with the notion of a vocabulary, which can be thought of as the URI references that are of interest in an OWL ontology. It is, however, not necessary that a vocabulary consist only of the URI references in an OWL ontology.

An OWL vocabulary V is a set of URI references, including http://www.w3.org/2002/07/owl#Thing (which will generally be written as owl:Thing) including http://www.w3.org/2002/07/owl#Nothing (which will generally be written as owl:Nothing). Each OWL vocabulary also includes URI references for a set of datatypes, D. Each datatype d in D is a 3-tuple $<L_d, V_d, L2V_d>$, where L_d is the lexical space of d, V_d is the value space for d, and $L2V_d$ is the lexical-to-value mapping for d, a mapping from L_d to V_d. If u is the canonical URI reference for an XML Schema datatype [*XML Schema Datatypes*] that is suitable for use in OWL then u is that datatype. In the semantics LV includes the (non-disjoint) union of the value spaces of the datatypes in D plus all Unicode strings.

An Abstract OWL interpretation with vocabulary V is a four-tuple of the form: I = <R, EC, ER, S> where

- R is a non-empty set of resources, disjoint from LV
- EC : V $\rightarrow$ 2^R $\cup$ 2^LV
- ER : V $\rightarrow$ 2^(R×R) $\cup$ 2^(R×LV)
- S : V $\rightarrow$ R

EC and ER provide meaning for URI references that are used as OWL classes and OWL properties, respectively. S provides meaning for URI references that are used to denote OWL individuals. S is extended to untyped literals by mapping them onto themselves, i.e., S(l) = l for l an untyped literal. S is extended to typed data literals by utilizing the L2V mapping as in S(d l) = $L2V_d$(l) for d l a typed literal.

Abstract OWL interpretations have the following conditions having to do with datatypes:

1. If d is the URI reference for a datatype then EC(d) = V_d.
2. If c is not the URI reference for any datatype then EC(c) $\subseteq$ R.

3.2. Interpreting Embedded Constructs

EC is extended to the syntactic constructs of descriptions, dataRanges, individuals, and propertyValues as in the Description Interpretation Table.

Description Interpretation Table

Abstract Syntax	Interpretation
http://www.w3.org/2002/07/owl#Thing (owl:Thing)	R
http://www.w3.org/2002/07/owl#Nothing (owl:Nothing)	{ }
complementOf(c)	R - EC(c)
unionOf(c_1 … c_n)	$EC(c_1) \cup \ldots \cup EC(c_n)$
intersectionOf(c_1 … c_n)	$EC(c_1) \cap \ldots \cap EC(c_n)$
oneOf(i_1 … i_n)	$\{S(i_1), \ldots, S(i_n)\}$
oneOf(v_1 … v_n)	$\{S(v_1), \ldots, S(v_n)\}$
restriction(p x_1 … x_n)	$EC(\text{restriction}(p\ x_1)) \cap \ldots \cap EC(\text{restriction}(p\ x_n))$
restriction(p allValuesFrom(r))	$\{x \in R \mid \langle x,y\rangle \in ER(p) \rightarrow y \in EC(r)\}$
restriction(p someValuesFrom(e))	$\{x \in R \mid \exists \langle x,y\rangle \in ER(p) \wedge y \in EC(e)\}$
restriction(p value(i))	$\{x \in R \mid \langle x,S(i)\rangle \in ER(p)\}$
restriction(p value(v))	$\{x \in R \mid \langle x,S(v)\rangle \in ER(p)\}$
restriction(p minCardinality(n))	$\{x \in R \mid card(\{y \in R \cup LV : \langle x,y\rangle \in ER(p)\}) \leq n\}$
restriction(p maxCardinality(n))	$\{x \in R \mid card(\{y \in R \cup LV : \langle x,y\rangle \in ER(p)\}) \geq n\}$
restriction(p cardinality(n))	$\{x \in R \mid card(\{y \in R \cup LV : \langle x,y\rangle \in ER(p)\}) = n\}$
Individual(annotation(…) … annotation(…) type(c_1) … type(c_m) pv_1 … pv_n)	$EC(c_1) \cap \ldots \cap EC(c_m) \cap EC(pv(pv_1)) \cap \ldots \cap EC(pv(pv_n))$
Individual(i annotation(…) … annotation(…) type(c_1) … type(c_m) pv_1 … pv_n)	$\{S(i)\} \cap EC(c) \cap \ldots \cap EC(c_m) \cap EC(pv(pv_1)) \cap \ldots \cap EC(pv(pv_n))$
value(p Individual(…))	$\{x \in R \mid \exists\ y \in EC(\text{Individual}(\ldots)) : \langle x,y\rangle \in ER(p)\}$
value(p id) for id an individualID	$\{x \in R \mid \langle x,S(id)\rangle \in ER(p)\ \}$
value(p v)	$\{x \in R \mid \langle x,S(v)\rangle \in ER(p)\ \}$

3.3. Interpreting Axioms and Facts

An Abstract OWL interpretation, I, satisfies OWL axioms and facts as given in Axiom Interpretation Table. In the table, optional parts of axioms and facts are given in square brackets ([…]) and have corresponding optional conditions, also given in square brackets.

Interpretation of Axioms and Facts

Directive	**Conditions on interpretations**
Class(c complete annotation(…) … annotation(…) $descr_1$ … $descr_n$)	$EC(c) = EC(descr_1) \cap \ldots \cap EC(descr_n)$
Class(c partial annotation(…) … annotation(…) $descr_1$ … $descr_n$)	$EC(c) \subseteq EC(descr_1) \cap \ldots \cap EC(descr_n)$
EnumeratedClass(c annotation(…) … annotation(…) i_1 … i_n)	$EC(c) = \{ S(i_1), \ldots, S(i_n) \}$
DisjointClasses(d_1 … d_n)	$EC(d_i) \cap EC(d_j) = \{ \}$ for $1 \le i < j \le n$
EquivalentClasses(d_1 … d_n)	$EC(d_i) = EC(d_j)$ for $1 \le i < j \le n$
SubClassOf(d_1 d_2)	$EC(d_1) \subseteq EC(d_2)$
DatatypeProperty(p annotation(…) … annotation(…) super(s_1) … super(s_n) domain(d_1) … domain(d_n) range(r_1) … range(r_n) [Functional])	$ER(p) \subseteq R \times LV \cap ER(s_1) \cap \ldots \cap ER(s_n) \cap EC(d_1) \times LV \cap \ldots \cap EC(d_n) \times LV \cap R \times EC(r_1) \cap \ldots \cap R \times EC(r_n)$ [ER(p) is functional]
ObjectProperty(p annotation(…) … annotation(…) super(s_1) … super(s_n) domain(d_1) … domain(d_n) range(r_1) … range(r_n) [inverse(i)] [Symmetric] [Functional] [InverseFunctional] [Transitive])	$ER(p) \subseteq R \times R \cap ER(s_1) \cap \ldots \cap ER(s_n) \cap EC(d_1) \times R \cap \ldots \cap EC(d_n) \times R \cap R \times EC(r_1) \cap \ldots \cap R \times EC(r_n)$ [ER(p) is the inverse of ER(i)] [ER(p) is symmetric] [ER(p) is functional] [ER(p) is inverse functional] [ER(p) is transitive]
EquivalentProperties(p_1 … p_n)	$ER(p_i) = ER(p_j)$ for $1 \le i < j \le n$
SubPropertyOf(p_1 p_2)	$ER(p_1) \subseteq ER(p_2)$
SameIndividual(i_1 … i_n)	$S(i_j) = S(i_k)$ for $1 \le j < k \le n$
DifferentIndividuals(i_1 … i_n)	$S(i_j) \neq S(i_k)$ for $1 \le j < k \le n$
Individual([i] annotation(…) … annotation(…) type(c_1) … type(c_m) pv_1 … pv_n)	EC(Individual([i] type(c_1) … type(c_m) pv_1 … pv_n)) is nonempty

3.4. Interpreting Ontologies

From Section 2, an OWL ontology in the abstract syntax is a sequence of axioms, facts, imports, and annotations.

The effect of an imports construct is to import the contents of another OWL ontology into the current ontology. The imported ontology is the one that can be found by accessing the document at the URI that is the argument of the imports construct. The *imports closure* of an OWL ontology is then the result of adding the contents of imported ontologies into the current ontology. If these contents contain further imports constructs, the process is repeated as necessary. A particular ontology is never imported more than once in this process, so loops can be handled. Annotation directives have no effect on the semantics of OWL ontologies in the abstract syntax.

Definitions: The top-level technical notion in this semantics for OWL is then whether an interpretation satisfies an OWL ontology, and the derived notion of entailment. An Abstract OWL interpretation, I, *satisfies* an OWL ontology, O, iff I satisfies each axiom and fact in the imports closure of O. An Abstract OWL ontology is *consistent* if there is some interpretation that satisfies it. An Abstract OWL ontology *entails* an OWL axiom or fact if each interpretation that satisfies the ontology also satisfies the axiom or fact. An Abstract OWL ontology *entails* another Abstract OWL ontology if each interpretation that satisfies the first ontology also satisfies the second ontology. Note that there is no need to create the imports closure of an ontology - any method that correctly determines the entailment relation is allowed.

---

4. Mapping to RDF Graphs

The exchange syntax for OWL is RDF/XML [*RDF Syntax*], as specified in the OWL Reference Description [*OWL Reference*]. Further, the meaning of an OWL ontology in RDF/XML is determined only from the RDF graph [*RDF Concepts*] that results from the RDF parsing of the RDF/XML document. Thus one way of translating an OWL ontology in abstract syntax form into the exchange syntax is by giving a transformation of each directive into a collection of triples. As all OWL Lite constructs are special cases of constructs in the full abstract syntax, transformations are only provided for the OWL DL versions.

The syntax for triples used here is the one used in the RDF Model Theory [*RDF MT*]. In this variant, qualified names are allowed. As detailed in the RDF Model Theory turn this syntax into the standard one, expand the qualified names into URI references in the standard RDF manner, by concatenating the namespace name with the local name. The only namespace prefixes used in the transformation are rdf, rdfs, xsd, and owl, which correspond to the namspaces http://www.w3.org/1999/02/22-rdf-syntax-ns#, http://www.w3.org/2000/01/rdf-schema#, http://www.w3.org/2001/XMLSchema#, and http://www.w3.org/2002/07/owl#, respectively.

4.1. Translation to RDF Graphs

The Transformation Table gives transformation rules that transform the abstract syntax to the OWL exchange syntax. In a few cases, notably for the DifferentIndividuals construct, there are different transformation rules. In such cases either rule can be chosen, resulting in a non-deterministic translation. This non-determinism is needed to allow the generation of more RDF Graphs.

The left column of the table gives a piece of syntax (S), the center column gives its transformation (T(S)), and the right column gives an identifier for the main node of the transformation (M(T(S))), but only for syntactic constructs that can occur as pieces of directives. Repeating components are listed using ellipses, as in description$_1$; … description$_n$, this form allows easy specification of the transformation for all values of n greater than or equal to zero. Optional portions of the abstract syntax (enclosed in square brackets) are optional portions of the transformation (signified by square brackets).

Some transformations in the table are for directives. Other transformations are for parts of directives. The last transformation is for sequences, which are not part of the abstract syntax per se. This last transformation is used to make some of the other transformations more compact and easier to read.

For many directives these transformation rules call for the transformation of components of the directive using other transformation rules. When the transformation of a component is used as the subject or object of a triple, the transformation of the construct is part of the production (but only once per production) and the main node of that transformation should be used for that node of the triple.

Bnode identifiers here must be taken as local to each transformation, i.e., different identifiers should be used for each invocation of a transformation rule. Also, some of the transformations require a URI for the current ontology. It is assumed that ontologies that are subject to this sort of transformation will be placed into a web-accessible document; the URI of this document is given as U.

Transformation to Triples

Abstract Syntax (and sequences) - S	Transformation - T(S)	Main Node - M(T(S))
Ontology(directive$_1$ … directive$_n$)	U rdf:type owl:Ontology . T(directive$_1$) … T(directive$_n$)	
Ontology(directive$_1$ … directive$_n$)	T(directive$_1$) … T(directive$_n$)	
datatypeID	datatypeID rdf:type rdfs:Datatype .	datatypeID
classID	classID rdf:type owl:Class .	classID
individualID		individualID
datavaluedPropertyID	datavaluedPropertyID rdf:type owl:DatatypeProperty .	datavalued-PropertyID
individualvaluedPropertyID	individualvaluedPropertyID rdf:type owl:ObjectProperty .	individualvalued-PropertyID
datatypeID literal	literal^^<datatypeID>	literal^^ <datatypeID>
literal	literal	literal
`Annotation`(URIreference URIreference)	U <URIreference> <URIreference> .	

Abstract Syntax (and sequences) - S	Transformation - T(S)	Main Node - M(T(S))
`Annotation`(URIreference dataLiteral)	U <URIreference> T(dataLiteral) .	
`Imports`(URI)	U owl:imports URI .	
`Individual`(iID annotation$_1$ … annotation$_n$ type(type$_1$)…type(type$_n$) value(pID$_1$ value$_1$) … value(pID$_n$ value$_n$))	iID T(annotation$_1$) . … iID T(annotation$_n$) . iID rdf:type T(type$_1$) . … iID rdf:type T(type$_n$) . iID pID$_1$ T(value$_1$) . … iID pID$_n$ T(value$_n$) .	iID
`Individual`(annotation$_1$ … annotation$_n$ type(type$_1$)…type(type$_n$) value(pID$_1$ value$_1$) … value(pID$_n$ value$_n$))	_:x T(annotation$_1$) . … _:x T(annotation$_n$) . _:x rdf:type T(type$_1$) . … _:x rdf:type T(type$_n$) . _:x pID$_1$ T(value$_1$) . … _:x pID$_n$ T(value$_n$) .	_:x
`SameIndividual`(iID$_1$ … iID$_n$)	iID$_i$ owl:sameIndividualAs iID$_j$. 1≤ij≤n	
`DifferentIndividuals`(iID$_1$ … iID$_n$)	iID$_i$ owl:differentFrom iID$_j$. 1≤ij≤n	
`DifferentIndividuals`(iID$_1$ … iID$_n$)	_:x rdf:type owl:AllDifferent . _:x owl:distinctMembers T(SEQ iID$_i$ … iID$_j$)	
`Class`(classID `partial` annotation$_1$ … annotation$_n$ description$_1$ … description$_n$)	classID rdf:type owl:Class . classID T(annotation$_1$) . … classID T(annotation$_n$) . classID rdfs:subClassOf T(description$_1$) . … classID rdfs:subClassOf T(description$_n$) .	
`Class`(classID `complete` annotation$_1$ … annotation$_n$ description)	classID rdf:type owl:Class . classID T(annotation$_1$) . … classID T(annotation$_n$) . classID owl:sameClassAs T(description) .	

Abstract Syntax (and sequences) - S	Transformation - T(S)	Main Node - M(T(S))
`Class`(classID `complete` annotation$_1$ … annotation$_n$ description$_1$ … description$_n$)	classID rdf:type owl:Class . classID T(annotation$_1$) . … classID T(annotation$_n$) . classID owl:intersectionOf T(SEQ description$_1$…description$_n$) .	
`EnumeratedClass`(classID annotation$_1$ … annotation$_n$ iID$_1$ … iID$_n$)	classID rdf:type owl:Class . classID T(annotation$_1$) . … classID T(annotation$_n$) . classID owl:oneOf T(SEQ iID$_1$…iID$_n$) .	
`DisjointClasses`(description$_1$ … description$_n$)	T(description$_i$) owl:disjointWith T(description$_j$) . 1≤ij≤n	
`EquivalentClasses`(description$_1$ … description$_n$)	T(description$_i$) owl:sameClassAs T(description$_j$) . 1≤ij≤n	
`SubClassOf`(description$_1$ description$_2$)	T(description$_1$) rdfs:subClassOf T(description$_2$) .	
`unionOf`(description$_1$ … description$_n$)	_:x owl:unionOf T(SEQ description$_1$…description$_n$) .	_:x
`intersectionOf`(description$_1$ … description$_n$)	_:x owl:intersectionOf T(SEQ description$_1$…description$_n$) .	_:x
`complementOf`(description)	_:x owl:complementOf T(description) .	_:x
`oneOf`(iID$_1$ … iID$_n$)	_:x owl:oneOf T(SEQ iID$_1$…iID$_n$) .	_:x
`oneOf`(v$_1$ … v$_n$)	_:x owl:oneOf T(SEQ v$_1$ … v$_n$) .	_:x
`restriction`(ID component$_1$ … component$_n$)	_:x owl:intersectionOf T(SEQ(restriction(ID component$_1$)… restriction(ID component$_n$))) .	_:x
`restriction`(ID allValuesFrom(range))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:allValuesFrom T(range) .	_:x

Abstract Syntax (and sequences) - S	Transformation - T(S)	Main Node - M(T(S))
`restriction`(ID someValuesFrom(required))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:someValuesFrom T(required) .	_:x
`restriction`(ID value(value))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:hasValue T(value) .	_:x
`restriction`(ID minCardinality(min))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:minCardinality "min"^^xsd:nonNegativeInteger .	_:x
`restriction`(ID maxCardinality(max))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:maxCardinality "max"^^xsd:nonNegativeInteger .	_:x
`restriction`(ID cardinality(card))	_:x rdf:type owl:Restriction . _:x owl:onProperty T(ID) . _:x owl:cardinality "card"^^xsd:nonNegativeInteger .	_:x
`DatatypeProperty`(ID annotation$_1$ … annotation$_n$ super(super$_1$)… super(super$_n$) domain(domain$_1$)… domain(domain$_n$) range(range$_1$)… range(range$_n$) [`Functional`])	ID rdf:type owl:DatatypeProperty . ID T(annotation$_1$) . … ID T(annotation$_n$) . ID rdfs:subPropertyOf T(super$_1$) . … ID rdfs:subPropertyOf T(super$_n$) . ID rdfs:domain T(domain$_1$) . … ID rdfs:domain T(domain$_n$) . ID rdfs:range T(range$_1$) . … ID rdfs:range T(range$_n$) . [ID rdf:type owl:FunctionalProperty .]	

Abstract Syntax (and sequences) - S	Transformation - T(S)	Main Node - M(T(S))
`ObjectProperty`(ID annotation$_1$ … annotation$_n$ super(super$_1$)… super(super$_n$) `domain`(domain$_1$)… `domain`(domain$_n$) `range`(range$_1$)… `range`(range$_n$) [`inverseOf`(inverse)] [`Symmetric`] [`Functional` | `InverseFunctional` | `Transitive`])	ID rdf:type owl:ObjectProperty . ID T(annotation$_1$) . … ID T(annotation$_n$) . ID rdfs:subPropertyOf T(super$_1$) . … ID rdfs:subPropertyOf T(super$_n$) . ID rdfs:domain T(domain$_1$) . … ID rdfs:domain T(domain$_n$) . ID rdfs:range T(range$_1$) . … ID rdfs:range T(range$_n$) . [ID owl:inverseOf T(inverse) .] [ID rdf:type owl:SymmetricProperty .] [ID rdf:type owl:FunctionalProperty .] [ID rdf:type owl:InverseFunctionalProperty .] [ID rdf:type owl:TransitiveProperty .]	
`EquivalentProperties`(dvpID$_1$ … dvpID$_n$)	T(dvpID$_i$) owl:samePropertyAs T(dvpID$_j$) . 1≤ij≤n	
`SubPropertyOf`(dvpID$_1$ dvpID$_2$)	T(dvpID$_1$) rdfs:subPropertyOf T(dvpID$_2$) .	
`EquivalentProperties`(ivpID$_1$ … ivpID$_n$)	T(ivpID$_i$) owl:samePropertyAs T(ivpID$_j$) . 1≤ij≤n	
`SubPropertyOf`(ivpID$_1$ ivpID$_2$)	T(ivpID$_1$) rdfs:subPropertyOf T(ivpID$_2$) .	
annotation (URIreference URIreference)	<URIreference> <URIreference>	
annotation (URIreference dataLiteral)	<URIreference> T(dataLiteral)	
SEQ		rdf:nil
SEQ item$_1$…item$_n$	_:l$_1$ rdf:type rdf:List . _:l$_1$ rdf:first T(item$_1$) . _:l$_1$ rdf:rest _:l$_2$. … _:ln rdf:type rdf:List . _:ln rdf:first T(item$_n$) . _:ln rdf:rest rdf:nil .	_:l$_1$

4.2. Definition of OWL DL and OWL Lite Ontologies in RDF Graph Form

Definition: An RDF graph is an OWL DL ontology in RDF graph form if it is equal (see below for a slight expansion) to a result of the transformation to triples above of the imports closure of an OWL DL ontology in abstract syntax form and, moreover,

1. the abstract syntax form does not use any URI reference as more than one of a classID, a datatypeID, an individualID, a datavaluedPropertyID, or an individualvaluedPropertyID;
2. the abstract syntax form does not use the URI of the document itself as a URI reference;
3. the abstract syntax form does not use any URI reference that is the first argument of an Annotation directive or an annotation portion of a directive as a classID, a datatypeID, an individualID, a datavaluedPropertyID, or an individualvaluedPropertyID; and
4. the abstract syntax form provides a type for every individualID;
5. the abstract syntax form does not mention any of the URI references from the RDF, RDFS, or OWL namespaces that are given special meaning in RDF, RDFS, or OWL except owl:Thing and owl:Nothing.

For the purposes of determining whether an RDF graph is an OWL DL ontology in RDF graph form, cardinality restrictions are explicitly allowed to use constructions like `"1"^^xsd:decimal` so long as the data value so encoded is a non-negative integer.

Definition: An RDF graph is an OWL Lite ontology in RDF graph form if it is equal (with the same relaxation as for OWL DL) to a result of the transformation to triples above of the imports closure of an OWL Lite ontology in abstract syntax form that meets the requirements given just above.

This transformation is not injective, as several OWL abstract ontologies that do not use the above reserved vocabulary can map into equal RDF graphs. However, the only cases where this can happen is with constructs that have the same meaning, such as several DisjointClasses axioms having the same effect as one larger one. It would be possible to define a canonical inverse transformation, if desired.

The above definition of OWL DL and OWL Lite ontologies in RDF graph form is couched from the perspective of the abstract syntax, as this is the perspective from with it can be easily stated. The following, much longer description of OWL DL ontologies in RDF graph form is couched from the perspective of RDF graphs. This description is strictly informative. The normative definition of what makes an OWL DL ontology in RDF graph form is given above.

Let G be an RDF graph. A node x1 in G is *a non-empty list in G* with elements e1,…,en if there is a set of triples in G of the form

```
x1 rdf:type rdf:List .
x1 rdf:first e1 .
x1 rdf:rest x2 .
...
xn rdf:type rdf:List .
xn rdf:first en .
xn rdf:rest rdf:nil .
```

where each xi is a distinct blank node that appears as the object of exactly one triple in G and x1 does not appear as the object of an rdf:rest triple. The definition triples of x1 are the triples above plus the definition triples of e1,...,en.

A node x in G is a *description* in G if x is a blank node and there is a set of triples of one of the entries in the Description Triples table, where

1. r is an object property or a datatype property in G.
2. if r is an object property in G then d is a description or a class; if r is a datatype property in G then d is a data range or a datatype.
3. if r is an object property in G then i is an individual and URI reference; if r is a datatype property in G then i is a typed or untyped literal.
4. n is a typed literal whose data value is a non-negative integer.
5. ds is rdf:nil or a non-empty list whose elements are descriptions or classes.
6. dc is a description or a class.
7. is is rdf:nil or a non-empty list whose elements are individuals that are also URI references.

Description Triples

Constructor	Triples

allValuesFrom	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:allValuesFrom d .`

someValuesFrom	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:someValuesFrom d .`

hasValue	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:hasValue i .`

minCardinality	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:minCardinality n .`

maxCardinality	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:maxCardinality n .`

cardinality	`x rdf:type owl:Restriction .`
	`x owl:onProperty r .`
	`x owl:cardinality n .`

unionOf	`x owl:unionOf ds .`
intersectionOf	`x owl:intersectionOf ds .`
complementOf	`x owl:complementOf dc .`
oneOf	`x owl:oneOf is .`

The definition triples of the description are the triples above plus the definition triples of any description, data range, or list in the triples above.

A node x in G is a *data range* in G if x is a blank node and there is a triple of the form `x owl:oneOf rdf:nil .` or `x owl:oneOf is .` where is is a non-empty list whose elements are typed or untyped literals. The definition triples of the data range are the triples above plus the definition triples of the list.

A node x in G is a *datatype property* if x is a URI reference and there is a triple of the form `x rdf:type owl:DatatypeProperty .` but no triple of any of the following forms

```
x rdf:type owl:SymmetricProperty .
x rdf:type owl:InverseFunctionalProperty .
x rdf:type owl:TransitiveProperty .
```

The assertions about x in G are the triples in G of the following forms, where y is a datatype property in G, d is a description or class in G, and r is a data range or datatype in G

```
x rdf:type owl:DatatypeProperty .
x rdfs:subPropertyOf y .
x rdfs:domain d .
x rdfs:range r .
x rdf:type owl:FunctionalProperty .
x owl:samePropertyAs y .
```

plus the definition triples of any description or data range in these triples.

A node x in G is an *object property* if x is a URI reference and there is a triple of the form `x rdf:type owl:ObjectProperty .` The assertions about x in G are the triples in G of the following forms, where y is an object property in G, d is a description or class in G

```
x rdf:type owl:ObjectProperty .
x rdfs:subPropertyOf y .
x rdfs:domain d .
x rdfs:range d .
x owl:inverseOf y .
x rdf:type owl:SymmetricProperty .
x rdf:type owl:FunctionalProperty .
x rdf:type owl:InverseFunctionalProperty .
x rdf:type owl:TransitiveProperty .
x owl:samePropertyAs y .
```

plus the definition triples of any description in these triples.

A node x in G is a *transitive object property* if x is an object property and there is a triple of the form `x rdf:type owl:TransitiveProperty .`

A node x in G is a *complex object property* if x is an object property and there is a triple of any of the following forms in G

```
x rdf:type owl:FunctionalProperty .
x rdf:type owl:InverseFunctionalProperty .
x owl:inverseOf y .
x owl:samePropertyAs y .
y rdfs:subClassOf x .
```

where y is a complex object property in G or if there are triples of each of the forms

```
z owl:onProperty x .
z c y .
```

where c is one of `owl:minCardinality`, `owl:maxCardinality`, or `owl:cardinality`.

A node x in G is a *datatype* if x is a URI reference and there is a triple of the form `x rdf:type owl:Datatype .` The assertions about x in G are the triples in G of the following forms, where d is a top-level description or class in G `x rdf:type owl:Datatype .`

A node x in G is a *class* if x is `owl:Thing` or `owl:Nothing` or x is a URI reference and there is a triple of the form `x rdf:type owl:Class .` The assertions about x in G are the triples in G of the following forms, where d is a top-level description or class in G and ds is rdf:nil or a non-empty list whose elements are descriptions or classes and is is rdf:nil or a non-empty list whose elements are individuals, plus the definition triples of any non-empty lists in these triples

```
x rdf:type owl:Class .
x rdfs:subClassOf d .
x owl:sameClassAs d .
x owl:disjointFrom d.
x owl:intersectionOf ds.
x owl:oneOf is .
```

A node x in G is a *top-level description* if x is a description that is not in the definition triples of any other description in G nor an element of a list in G. The assertions about x in G are the triples in G of the following forms, where d is a top-level description or class in G

```
x rdf:type owl:Class .
x rdfs:subClassOf d .
x owl:sameClassAs d .
x owl:disjointWith d .
```

plus the definition triples of x.

A node x in G is an *individual* if there is a triple of the form `x rdf:type c .` where c is a description or class. The assertions about x in G are the triples of G of the following forms, where c is a description or class, rd is a datatype property, v is a typed or untyped literal, ro is an object property, and i is an individual.

```
x rdf:type c .
x rd v .
x ro i .
x owl:sameIndividualAs i .
x owl:differentFrom i .
```

plus the definition triples of any description in these triples.

A node x in G is an *all-different node* if there is a triple of the form `x rdf:type owl:allDifferent .` and there is exactly one other triple in G whose subject is x and this triple is of one of the following forms, where l is a non-empty list whose elements are individuals

```
x owl:distinctMembers owl:nil .
x owl:distinctMembers l .
```

The assertions about x are the above triples, plus the definition triples of l, if present.

A node x in G is an *ontology* if x is a URI reference and there is a triple of the form `x rdf:type owl:Ontology .` The assertions about x are the triples of G of the following form

```
x rdf:type owl:Ontology .
x owl:imports y .
```

where y is a URI reference that is not a datatype property, an object property, a class, or an individual.

An RDF graph is an OWL DL graph if:

1. The datatype properties of G, the object properties of G, the classes of G, the datatypes of G, the individuals of G, and the ontologies of G are pairwise disjoint and disjoint from the RDF, RDFS, and OWL vocabularies (except owl:Thing, owl:Nothing, rdfs:Literal, and rdf:XMLLiteral).
2. The definition triples of any description or datarange are disjoint from the definition triples of other description or data range except for any description or data range that is a node of some triple in the definition triples of the description or data range. (Therefore the definition triples of any description or datarange form a tree.)
3. The triples of the graph can be disjointly partitioned such that each partition is either
    1. the assertions about a datatype property, object property, datatype, class, top-level description, individual, all-different node, or ontology;
    2. (annotation) triples whose subject is a datatype property, an object property, a datatype, a class, or an individual and whose predicate is not a datatype property, an object property, a class, an individual, nor any URI reference from the RDF, RDFS, or OWL namespaces that is given special meaning by RDF, RDFS, or OWL; or
    3. (ontology annotation) triples whose subject is an ontology and whose predicate is not a datatype property, an object property, a datatype, a class, an individual, or any URI reference from the RDF, RDFS, or OWL namespaces that is given special meaning by RDF, RDFS, or OWL.
4. The complex object properties and the transitive object properties of G are disjoint.

A quick incomplete gloss of the above is that

1. In an OWL DL ontology in RDF graph form a resource cannot be more than one of a class, a datatype, an object property, a datatype property, or an individual. OWL DL requires that inverse functional properties, symmetric properties, and transitive properties be object properties, so they cannot be datatype properties.
2. In an OWL DL ontology in RDF graph form an object property that participates in a cardinality restriction cannot be specified as a transitive property nor can it have a transitively-specified property as a descendant.
3. In an OWL DL ontology in RDF graph form all descriptions must be well-formed, with no missing or extra components, and must form tree-like structures.

---

5. RDFS-Compatible Model-Theoretic Semantics

This model-theoretic semantics for OWL is an extension of the RDFS semantics as defined in the RDF model theory [*RDF MT*], and defines the OWL semantic extension of RDF.

5.1. The OWL and RDFS universes

All of the OWL vocabulary is defined on the 'OWL universe', which is a collection of RDFS classes that are intended to circumscribe the domain of application of the OWL vocabulary: owl:Thing, owl:Class and owl:Property. The class extension of owl:Thing comprises the individuals of the OWL universe. The class extension of owl:Class comprises the classes of the OWL universe. The class extension of owl:Property comprises the properties of the OWL universe.

There are two different styles of using OWL. In the more free-wheeling style, called OWL Full, the three domain-circumscription classes, owl:Thing, owl:Class and owl:Property, are identified with their RDFS counterparts. In OWL Full, as in RDFS, resources can be both an individual and a class, or, in fact, even an individual, a class, and a property. In the more restrictive style, called OWL DL here, the three domain-circumscription classes are different from their RDFS counterparts and, moreover, pairwise disjoint.

The OWL DL style gives up some expressive power in return for decidability of entailment. Both styles of OWL provide entailments that are missing in a naive translation of the DAML+OIL model-theoretic semantics into the RDFS semantics.

The chief differences in practice between the two styles lie in the care that is required to ensure that URI references are actually in the appropriate OWL universe. In OWL Full, no care is needed, as the OWL universe is the same as the RDFS universe. In OWL DL, localizing information must be provided for many of the URI references used. These localizing assumptions are all trivially true in OWL Full, and can also be ignored when one uses the OWL abstract syntax. But when writing OWL DL in triples close attention must be paid to which parts of the vocabulary can be 'legally' used to refer to things in the OWL universe.

5.1.1. Qualified Names and URI References

Throughout this section qualified names are used as shorthand for URI references. The namespace identifiers used in such names, namely rdf, rdfs, xsd, and owl, should be used as if they are given their usual definitions. Throughout this section VRDFS is the RDF and RDFS built-in vocabulary, i.e., rdf:type, rdf:Property, rdfs:Class, rdfs:subClassOf, …, minus rdfs:Literal; and VOWL is the OWL built-in vocabulary, i.e., owl:Class, owl:onProperty, …, minus owl:Thing and owl:Nothing.

5.2. OWL Interpretations

The semantics of OWL DL and OWL Full are very similar. The common portion of their semantics is thus given first, and the differences left until later.

From the RDF model theory [*RDF MT*], for V a set of URI references containing the RDF and RDFS vocabulary, an RDFS interpretation over V is a triple $I = < R_I, EXT_I, S_I >$, where $LV \subseteq R_I$. Here R_I is the domain of discourse or universe, i.e., a set that contains the denotations of URI references. EXT_I is used to give meaning to properties, and is a mapping from R_I to sets of pairs over $R_I \times R_I$. Finally, S_I is a mapping from V to R_I that takes a URI reference to its denotation. $CEXT_I$ is then defined as $CEXT_I(c) = \{ x \in R_I \mid <x,c> \in EXT_I(S_I(\text{rdf:type})) \}$. RDFS interpretations must meet several conditions, as detailed

in the RDFS model theory. For example, I(rdfs:subClassOf) must be a transitive relation.

An OWL interpretation, $I = < R_I, EXT_I, S_I, L_I >$, over a vocabulary V, where V includes VRDFS, rdfs:Literal, VOWL, owl:Thing, and owl:Nothing, is a datatyped RDFS interpretation over D, a set of datatypes including all the useful XML Schema datatypes simple types in Section 2 and V that satisfies all the constraints in this section.

The OWL vocabulary that corresponds to Description Logic constructors are given a different treatment from the OWL vocabulary that corresponds to (other) semantic relationships. The former have only-if semantic conditions and comprehension principles; the latter have if-and-only-if semantic conditions. The only-if semantic conditions for the former are needed to prevent semantic paradoxes and other problems with the semantics. The comprehension principles for the former and the if-and-only-if semantic conditions for the latter are needed so that useful entailments are valid.

If-and-only-if conditions for RDFS domains and ranges

If E is	**then for**	**$<x,y> \in EXT_I(S_I(E))$ iff**
rdfs:domain	$x \in IOP, y \in IOC$	$<z,w> \in EXT_I(x) \rightarrow z \in CEXT_I(y)$
rdfs:range	$x \in IOP, y \in IOC \cup IDC$	$<w,z> \in EXT_I(x) \rightarrow z \in CEXT_I(y)$

Conditions concerning the OWL universe and syntactic categories

If E is	then		
	$S_I(E) \in$	$CEXT_I(S_I(E)) =$	and
owl:Class		IOC	$IOC \subseteq CEXT_I(S_I(rdfs:Class))$
owl:Thing	IOC	IOT	$IOT \subseteq R_I$
owl:Nothing	IOC	{ }	
owl:Restriction		IOR	$IOR \subseteq IOC$
owl:Property		IOP	$IOP \subseteq CEXT_I(S_I(rdf:Property))$
owl:ObjectProperty		IOOP	$IOOP \subseteq IOP$
owl:DatatypeProperty		IODP	$IODP \subseteq IOP$
rdfs:Datatype		IDC	$IDC \subseteq CEXT_I(S_I(rdfs:Class))$
rdfs:Literal	IDC	LV	
owl:Ontology			
owl:AllDifferent		IAD	
rdf:List		IL	$IL \subseteq R_I$
rdf:nil	IL		

Characteristics of OWL classes, datatypes, and properties

If E is	then if $e \in CEXT_I(S_I(E))$ then
owl:Class	$CEXT_I(e) \subseteq IOT$
rdfs:Datatype	$CEXT_I(e) \subseteq LV$
owl:ObjectProperty	$EXT_I(e) \subseteq IOT \times IOT$
owl:DatatypeProperty	$EXT_I(e) \subseteq IOT \times LV$
If E is	**then $c \in CEXT_I(S_I(E))$ iff $c \in IOP$ and**
owl:SymmetricProperty	$\langle x,y \rangle \in EXT_I(c) \rightarrow \langle y, x \rangle \in EXT_I(c)$
owl:FunctionalProperty	$\langle x,y_1 \rangle, \langle x,y_2 \rangle \in EXT_I(c) \rightarrow y_1 = y_2$
owl:InverseFunctionalProperty	$\langle x_1,y \rangle, \langle x_2,y \rangle \in EXT_I(c) \rightarrow x_1 = x_2$
owl:TransitiveProperty	$\langle x,y \rangle, \langle y,z \rangle \in EXT_I(c) \rightarrow \langle x,z \rangle \in EXT_I(c)$

Characteristics of OWL vocabulary related to equivalence

If E is	then $\langle x,y\rangle \in EXT_I(S_I(E))$ iff
owl:sameClassAs	$x,y \in IOC$ and $CEXT_I(x)=CEXT_I(y)$
owl:disjointWith	$x,y \in IOC$ and $CEXT_I(x) \cap CEXT_I(y)=\{\}$
owl:samePropertyAs	$x,y \in IOP$ and $EXT_I(x) = EXT_I(y)$
owl:inverseOf	$x,y \in IOOP$ and $\langle u,v\rangle \in EXT_I(x)$ iff $\langle v,u\rangle \in EXT_I(y)$
owl:sameIndividualAs	$x = y$
owl:sameAs	$x = y$
owl:differentFrom	$x \neq y$

Conditions on OWL vocabulary related to boolean combinations and sets

We will say that l_1 is a sequence of $y_1,\ldots,y_n$ over C iff n=0 and $l_1=S_I$(rdf:nil) or n>0 and $l_1 \in IL$ and $\exists$ $l_2, \ldots, l_n \in IL$ such that
$\langle l_1,y_1\rangle \in EXT_I(S_I(\text{rdf:first}))$, $y_1 \in CEXT_I(C)$, $\langle l_1,l_2\rangle \in EXT_I(S_I(\text{rdf:rest}))$, …,
$\langle l_n,y_n\rangle \in EXT_I(S_I(\text{rdf:first}))$, $y_n \in CEXT_I(C)$, and $\langle l_n,S_I(\text{rdf:nil})\rangle \in EXT_I(S_I(\text{rdf:rest}))$.

If E is	then if $\langle x,y\rangle \in EXT_I(S_I(E))$ then
owl:complementOf	$x,y \in IOC$ and $CEXT_I(x)=IOT-CEXT_I(y)$
If E is	**then if $\langle x,l\rangle \in EXT_I(S_I(E))$ then**
owl:unionOf	$x \in IOC$ and l is a sequence of $y_1,\ldots y_n$ over IOC and $CEXT_I(x) = CEXT_I(y_1) \cup \ldots \cup CEXT_I(y_n)$
owl:intersectionOf	$x \in IOC$ and l is a sequence of $y_1,\ldots y_n$ over IOC and $CEXT_I(x) = CEXT_I(y_1) \cap \ldots \cap CEXT_I(y_n)$
owl:oneOf	$x \in CEXT_I(S_I(\text{rdfs:Class}))$ and l is a sequence of $y_1,\ldots y_n$ over IOT or over LV and $CEXT_I(x) = \{y_1,\ldots, y_n\}$

Further conditions on owl:oneOf

If E is	and	then if $\langle x,l\rangle \in EXT_I(S_I(E))$ then
owl:oneOf	l is a sequence of $y_1,\ldots y_n$ over LV	$x \in IDC$ and $CEXT_I(x) = \{y_1,...., y_n\}$
owl:oneOf	l is a sequence of $y_1,\ldots y_n$ over IOT	$x \in IOC$ and $CEXT_I(x) = \{y_1,...., y_n\}$

Conditions on OWL restrictions

If	then $x \in IOR$, $y \in IOC \cup IDC$, $p \in IOP$, and $CEXT_I(x) =$
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:allValuesFrom}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid \langle u,v\rangle \in EXT_I(p) \rightarrow v \in CEXT_I(y)\}$
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:someValuesFrom}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid \exists \langle u,v\rangle \in EXT_I(p) \wedge v \in CEXT_I(y)\}$
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:hasValue}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid \langle u, y\rangle \in EXT_I(p)\}$
If	**then $x \in IOR$, $y \in LV$, y is a non-negative integer, $p \in IOP$, and $CEXT_I(x) =$**
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:minCardinality}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid card(\{v : \langle u,v\rangle \in EXT_I(p)\}) \geq y\}$
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:maxCardinality}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid card(\{v : \langle u,v\rangle \in EXT_I(p)\}) \leq y\}$
$\langle x,y\rangle \in EXT_I(S_I(\text{owl:cardinality}))) \wedge \langle x,p\rangle \in EXT_I(S_I(\text{owl:onProperty})))$	$\{u \in IOT \mid card(\{v : \langle u,v\rangle \in EXT_I(p)\}) = y\}$

Comprehension conditions (principles)

The first two comprehension conditions require the existence of the finite sequences that are used in some OWL constructs. The third comprehension condition requires the existence of instances of owl:AllDifferent. The remaining comprehension conditions require the existence of the appropriate OWL descriptions and data ranges.

If there exists	then there exists $l_1,\ldots,l_n \in IL$ with
$x_1, \ldots, x_n \in IOC$	$\langle l_1,x_1\rangle \in EXT_I(S_I(\text{rdf:first}))$, $\langle l_1,l_2\rangle \in EXT_I(S_I(\text{rdf:rest}))$, … $\langle l_n,x_n\rangle \in EXT_I(S_I(\text{rdf:first}))$, $\langle l_n,S_I(\text{rdf:nil})\rangle \in EXT_I(S_I(\text{rdf:rest}))$

If there exists	**then there exists $l_1,\ldots,l_n \in$ IL with**
$x_1, \ldots, x_n \in$ IOT$\cup$LV	$\langle l_1,x_1\rangle \in EXT_I(S_I(\text{rdf:first}))$, $\langle l_1,l_2\rangle \in EXT_I(S_I(\text{rdf:rest}))$, … $\langle l_n,x_n\rangle \in EXT_I(S_I(\text{rdf:first}))$, $\langle l_n,S_I(\text{rdf:nil})\rangle \in EXT_I(S_I(\text{rdf:rest}))$
If there exists	**then there exists y with**
l, a sequence of $x_1,\ldots,x_n$ over IOC with $x_i \neq x_j$ for $1\leq i<j\leq n$	$y \in$ IAD, $\langle y,l\rangle \in EXT_I(S_I(\text{owl:distinctMembers}))$
If there exists	**then there exists y with**
l, a sequence of $x_1,\ldots,x_n$ over IOC	$y \in$ IOC, $\langle y,l\rangle \in EXT_I(S_I(\text{owl:unionOf}))$
l, a sequence of $x_1,\ldots,x_n$ over IOC	$y \in$ IOC, $\langle y,l\rangle \in EXT_I(S_I(\text{owl:intersectionOf}))$
l, a sequence of $x_1,\ldots,x_n$ over IOT	$y \in$ IOC, $\langle y,l\rangle \in EXT_I(S_I(\text{owl:oneOf}))$
l, a sequence of $x_1,\ldots,x_n$ over LV	$y \in$ IDC, $\langle y,l\rangle \in EXT_I(S_I(\text{owl:oneOf}))$
If there exists	**then there exists y $\in$ IOC with**
$x \in$ IOC	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:complementOf}))$
If there exists	**then there exists y $\in$ IOR with**
$x \in$ IOP $\wedge$ $w \in$ IOC $\cup$ IDC	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:allValuesFrom}))$
$x \in$ IOP $\wedge$ $w \in$ IOC $\cup$ IDC	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:someValuesFrom}))$
$x \in$ IOP $\wedge$ $w \in$ IOT $\cup$ LV	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:hasValue}))$
$x \in$ IOP $\wedge$ $w \in$ LV $\wedge$ w is a non-negative integer	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:minCardinality}))$
$x \in$ IOP $\wedge$ $w \in$ LV $\wedge$ w is a non-negative integer	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:maxCardinality}))$
$x \in$ IOP $\wedge$ $w \in$ LV $\wedge$ w is a non-negative integer	$\langle y,x\rangle \in EXT_I(S_I(\text{owl:onProperty})) \wedge$ $\langle y,w\rangle \in EXT_I(S_I(\text{owl:cardinality}))$

5.3. OWL DL

OWL DL augments the above conditions with a separation of the domain of discourse into several disjoint parts. This separation has two consequences. First, the OWL portion of the domain of discourse becomes standard first-order, in that predicates (classes and properties) and individuals are disjoint. Second, the OWL portion of a OWL DL interpretation can be viewed as a Description Logic interpretation for a particular expressive Description Logic.

Definition: A *OWL DL interpretation* over a vocabulary V, where V includes VRDFS, rdfs:Literal, VOWL, owl:Thing, and owl:Nothing, is an OWL interpretation as above that satisfies the following conditions.

The domain of discourse is divided up into several pieces.

LV, IOT, IOC, IDC, IOP, and IL are all pairwise disjoint.
There is a disjoint partition of IOP into IOOP and IODP.
For $n \in$ VRDFS$\cup$VOWL - {rdf:nil}, $S_I(n) \in R_I$ - (LV$\cup$IOT$\cup$IOC$\cup$IDC$\cup$IOP$\cup$IL).

5.3.1. OWL DL Entailment

Now entailment in OWL DL can be defined.

Definition: Let K be an RDF graph and let V be a vocabulary that includes VRDFS, rdfs:Literal, VOWL, owl:Thing, and owl:Nothing. An *OWL DL interpretation of K* is an OWL DL interpretation over V that is also an RDFS interpretation of K.

Definition: Let K be an RDF Graph. The *imports closure* of K is the smallest superset of K such that if the imports closure of K contains a triple of the form x owl:imports y . where x and y are any URIs then the imports closure of K contains the triples resulting from the RDF parsing of the document, if any, accessible at y into triples.

Definitions: Let K and Q be RDF Graphs and let V be a vocabulary that includes VRDFS, rdfs:Literal, VOWL, owl:Thing, and owl:Nothing. Then *K OWL DL entails Q* whenever all OWL DL interpretations of the RDF graph specified by imports closure of K are also OWL DL interpretations of the RDF graph specified by imports closure of Q. K is *OWL DL consistent* if there is some OWL DL interpretation of the imports closure of K.

5.3.2. Correspondence to the Direct Semantics

One way to automatically obtain typing information for the vocabulary is to use translations into RDF Graphs of certain kinds of OWL ontologies in the abstract syntax, as such translations contain information on what a URI reference denotes. This is made more formal below.

Definition: A *separated OWL vocabulary*, V', is a set of URI references with a disjoint partition V' = < VI, VC, VD, VOP, VDP >, where owl:Thing and owl:Nothing are in VC, rdfs:Literal is in VD, and all the elements of D are in VD. Further V' is disjoint from VRDFS∪VOWL.

Definition: An *OWL abstract ontology with separated names* over a separated OWL vocabulary V' = < VI, VC, VD, VOP, VDP > is a set of OWL axioms and facts in the abstract syntax without annotations as in Section 2 where <individualID>s are taken from VI, <classID>s are taken from VC, <datatypeID>s are taken from VD, <individualvaluedPropertyID>s are taken from VOP, and <datavaluedPropertyID>s are taken from VDP.

Theorem 1: Let T be the mapping from OWL ontologies in the abstract syntax to RDF Graphs. Let V' = < VI, VC, VD, VOP, VDP > be a separated OWL vocabulary. Let K and Q be OWL abstract syntax ontologies with separated names over V' and let V = V' ∪ VRDFS ∪ VOWL. Then it is the case that K entails Q if and only if T(K) OWL DL entails T(Q). The proof is contained in Appendix A.1.

A simple corollary of this is that K is consistent if and only if T(K) is consistent.

5.4. OWL Full

OWL Full augments the common conditions with conditions that force the OWL and RDFS universes to be the same.

A OWL Full interpretation over a vocabulary V, where V includes VRDFS, rdfs:Literal, VOWL, owl:Thing, and owl:Nothing, is an OWL interpretation as above that satisfies the following conditions.

$IOT = R_I$
$IOC = CEXT_I(S_I(\text{rdfs:Class}))$
$IOP = CEXT_I(S_I(\text{rdf:Property}))$

OWL Full entailment and *OWL Full consistency* are defined in same manner as OWL DL entailment and OWL DL consistency.

Theorem 2: Let K,C be RDF graphs such that each of K, C, and K∪C is the translation of some OWL ontology in the abstract syntax with separated vocabulary. Then K OWL Full entails C if K OWL DL entails C. An initial sketch of the proof is contained in Appendix A.2.

Appendix A. Proofs (Informative)

This appendix contains proofs of theorems contained in Section 5 of the document.

A.1 Correspondence between Abstract OWL and OWL DL

This section shows that the two semantics for OWL DL correspond on certain OWL ontologies. One semantics used in this section is the direct model theory for abstract OWL ontologies given in the direct model-theoretic semantics section of this document. The other semantics is the extension of the RDFS semantics given in the RDFS-compatible model-theoretic semantics section of this document.

Throughout this section qualified names are used as shorthand for URI references. The namespace identifiers used in such names, namely rdf, rdfs, xsd, and owl, should be used as if they are given their usual definitions.
Throughout this section VRDFS is the RDF and RDFS built-in vocabulary, i.e., rdf:type, rdf:Property, rdfs:Class, rdfs:subClassOf, …, minus rdfs:Literal; and VOWL is the OWL built-in vocabulary, i.e., owl:Class, owl:onProperty, …, minus owl:Thing and owl:Nothing.
Throughout this section D will be a datatyping scheme, i.e., a set of URI references that have class extensions that are subsets of LV and mappings from strings to their class extension.
Throughout this section T will be the mapping from abstract OWL ontologies to RDF Graphs.

A separated OWL vocabulary is a set of URI references V with a disjoint partition V = <VI, VC, VD, VOP, VDP>, where owl:Thing and owl:Nothing are in VC, rdfs:Literal is in VD, and all the elements of D are in VD. Further V is disjoint from VRDFS∪VOWL.

An OWL abstract KB with separated names over a separated OWL vocabulary V = <VI, VC, VD, VOP, VDP> is a set of OWL axioms and facts without annotations as in Section 2 where <individualID>s are taken from VI, <classID>s are taken from VC, <datatypeID>s are taken from VD, <individualvaluedPropertyIDs> are taken from VOP, and <datavaluedPropertyID>s are taken from VDP.

Let V = <VI, VC, VD, VOP, VDP> be a separated OWL vocabulary. Then T(V) is the RDF Graph that contains exactly <v,rdf:type,owl:Thing > for v ∈ VI, <v,rdf:type,owl:Class > for v ∈ VC, <v,rdf:type,rdfs:Datatype > for v ∈ VD, <v,rdf:type,owl:ObjectProperty > for v ∈ VOP, and <v,rdf:type,owl:DatatypeProperty > for v ∈ VDP.

The theorem to be proved is:
Let V' be a separated OWL vocabulary. Let K,Q be abstract OWL ontologies with separated names over V'. Then K OWL entails Q iff T(K),T(V') OWL DL entails T(Q).

Actually, a slightly stronger correspondence can be shown, but this is enough for now, as the presence of annotations and imports causes even more complications.

A.1.1 Lemma 1

Lemma 1: Let V' = <VI, VC, VD, VOP, VDP> be a separated OWL vocabulary.
Let V = VI ∪ VC ∪ VD ∪ VOP ∪ VDP ∪ VRDFS ∪ VOWL.
Let I'= <R,EC,ER,S> be an abstract OWL interpretation over V'.
Let I = <R_I,EXT_I,S_I,L_I> be an OWL DL interpretation over V of T(V'). Let $CEXT_I$ have its usual meaning.
If R=$CEXT_I$(I(owl:Thing)), S(v)=S_I(v) for v ∈ VI, EC(v)=$CEXT_I$(S_I(v)) for v∈ VC∪VD, and ER(v)=EXT_I(S_I(v)) for v∈ VOP∪VDP

then for d any abstract OWL description or data range over V', I OWL DL satisfies T(d) and for any E mapping all the blank nodes of T(d) into R_I where I+E OWL DL satisfies T(d),

1. $CEXT_I(I(M(T(d)))) = EC(d)$,
2. if d is a description then $I(M(T(d))) \in CEXT_I(I(owl:Class))$, and
3. if d is a data range then $I(M(T(d))) \in CEXT_I(I(rdfs:Datatype))$.

Proof

The proof of the lemma is by a structural induction. Throughout the proof, let IOT = $CEXT_I(I(owl:Thing))$, IOC = $CEXT_I(I(owl:Class))$, IDC = $CEXT_I(I(rdfs:Datatype))$, IOOP = $CEXT_I(I(owl:ObjectProperty))$, IODP = $CEXT_I(I(owl:DatatypeProperty))$, and IL = $CEXT_I(I(rdf:List))$.

To make the induction work, it is necessary to show that for any d a description or data range with sub-constructs T(d) contains triples for each of the sub-constructs that do not share any blank nodes with triples from the other sub-constructs. This can easily be verified from the rules for T.

If $p \in VOP$ then I satisfies $p \in IOOP$. Then, as I is an OWL DL interpretation, I satisfies $<p,I(owl:Thing)> \in EXT_I(I(rdfs:domain))$ and $<p,I(owl:Thing)> \in EXT_I(I(rdfs:range))$. Thus I satisfies T(p). Similarly for $p \in VDP$.

Base Case: $v \in$ VC, including owl:Thing and owl:Nothing
As $v \in VC$ and I satisfies T(V), thus $I(v) \in CEXT_I(I(owl:Class))$. Because I is an OWL DL interpretation $CEXT_I(I(v)) \subseteq IOT$, so $<I(v),I(owl:Thing)> \in EXT_I(I(rdfs:subClassOf))$. Thus I OWL DL satisfies T(v). As M(T(v)) is v, thus $CEXT_I(I(M(T(v)))) = EC(v)$. Finally, from above, $I(v) \in IOC$.

Base Case: $v \in$ VD, including rdfs:Literal
As $v \in VD$ and I satisfies T(V), thus $I(v) \in CEXT_I(I(rdfs:Datatype))$. Because I is an OWL DL interpretation $CEXT_I(I(v)) \subseteq LV$, so $<I(v),I(rdfs:Literal)> \in EXT_I(I(rdfs:subClassOf))$. Thus I OWL DL satisfies T(v). As M(T(v)) is v, thus $CEXT_I(I(M(T(v)))) = EC(v)$. Finally, from above $I(v) \in IDC$.

Base Case: $d=oneOf(i_1 \ldots i_n)$, where the i_j are individual IDs
As $i_j \in VI$ for $1 \le j \le n$ and I satisfies T(V), thus $I(i_j) \in IOT$. The second comprehension principle for sequences then requires that there is some $l \in IL$ that is a sequence of $I(i_1),\ldots,I(i_n)$ over IOT. For any l that is a sequence of $I(i_1),\ldots,I(i_n)$ over IOT the comprehension principle for oneOf requires that there is some $y \in CEXT_I(I(rdfs:Class))$ such that $<y,l> \in EXT_I(IS(owl:oneOf))$. From the third characterization of oneOf, $y \in IOC$. Therefore I satisfies T(d). For any I+E that satisfies T(d), $CEXT_I(I+E(M(T(d)))) = \{I(i_1),\ldots,I(i_n)\} = EC(d)$. Finally, $I+E(M(T(d))) \in IOC$.

Base Case: $d=oneOf(v_1 \ldots v_n)$, where the vi are typed data values
As v_j is an abstract syntax typed data value for $1 \le j \le n$ $I(v_j) \in LV$. The second comprehension principle for sequences then requires that there is some $l \in IL$ that is a sequence of $I(v_1),\ldots,I(v_n)$ over LV. For any l that is a sequence of $I(v_1),\ldots,I(v_n)$ over LV the comprehension principle for oneOf requires that there is some $y \in CEXT_I(I(rdfs:Class))$ such that $<y,l> \in EXT_I(IS(owl:oneOf))$. From the second characterization of oneOf, $y \in IOC$. Therefore I satisfies T(d). For any I+E that satisfies T(d), $CEXT_I(I+E(M(T(d)))) = \{I(i_1),\ldots,I(i_n)\} = EC(d)$. Finally, $I+E(M(T(d))) \in IDC$.

Base Case: d=restriction(p value(i)), with p∈ VOP∪VDP and i an individualID

As p∈ VOP∪VDP, from above I satisfies T(p). As I satisfies T(V'), I(p)∈ IOOP∪IODP. As i∈ VI and I satisfies T(V'), I(i)∈ IOT. From a comprehension principle for restriction, I satisfies T(d). For any E such that I+E satisfies T(d), $CEXT_I(I+E(M(T(d))))$ = { x∈ IOT | <x,I(i)> ∈ $EXT_I(p)$ } = { x∈ R | <x,S(i)> ∈ ER(p) } = EC(d). Finally, I+E(M(T(d)))∈ IOC.

Base Case: d=restriction(p value(i)), with p∈ VOP∪VDP and i a typed data value.

Similar.

Base Case: d=restriction(p minCardinality(n)), with p∈ VOP∪VDP and n a non-negative integer

Similar.

Base Case: d=restriction(p maxCardinality(n)), with p∈ VOP∪VDP and n a non-negative integer

Similar.

Base Case: d=restriction(p Cardinality(n)), with p∈ VOP∪VDP and n a non-negative integer

Similar.

Inductive Case: d=complementOf(d')

From the induction hypothesis, I satisfies T(d'). As d' is a description, from the induction hypothesis any extension of I, I+E, that satisfies T(d') has has I+E(M(T(d'))) = EC(d') and I+E(M(T(d')))∈ IOC. The comprehension principle for complementOf then requires that there is a y∈ IOC such that I+E satisfies <y,e>∈ EXT_I(I(owl:complementOf)) so I satisfies T(d). For any I+E that satisfies T(d), $CEXT_I(I+E(M(T(d))))$ = IOT-$CEXT_I(I+E(M(T(d))))$ = R-EC(d') = EC(d). Finally, I+E(M(T(d)))∈ IOC.

Inductive Case: d = unionOf(d_1 … d_n)

From the induction hypothesis, I satisfies d_i for $1 \le i \le n$ so there is an extension of I, E_i, that maps all the blank nodes in $T(d_i)$ into domain elements such that $I+E_i$ satisfies $T(d_i)$. As the blank nodes in $T(d_i)$ are disjoint from the blank nodes of $T(d_j)$ for i≠j, $I+E_1+\ldots+E_n$, and thus I, satisfies $T(d_i)\cup\ldots\cup T(d_n)$. Each d_i is a description, so from the induction hypothesis, $I+E_1+\ldots+E_n(M(T(d_i)))$∈ IOC. The first comprehension principle for sequences then requires that there is some l∈ IL that is a sequence of $I+E_1+\ldots+E_n(M(T(d_1)))$,…, $I+E_1+\ldots+E_n(M(T(d_n)))$ over IOC. The comprehension principle for unionOf then requires that there is some y∈ IOC such that <y,l>∈ EXT_I(I(owl:unionOf)) so I satisfies T(d).

For any I+E that satisfies T(d), I+E satisfies $T(d_i)$ so $CEXT_I(I+E(d_i)) = EC(d_i))$. Then $CEXT_I(I+E(M(T(d))))$ = $CEXT_I(I+E(d_1))\cup\ldots\cup CEXT_I(I+E(d_n))$ = $EC(d_1)\cup\ldots\cup EC(d_n)$ = EC(d). Finally, I(M(T(d)))∈ IOC.

Inductive Case: d = intersectionOf(d_1 … d_n)

Similar.

Inductive Case: d = restriction(p x_1 x_2 … x_n)

As p∈ VOP∪VDP, from above I satisfies T(p). From the induction hypothesis, I satisfies restriction(p x_i) for $1 \le i \le n$ so there is an extension of I, E_i, that maps all the blank nodes in T(restriction(p x_i)) into domain elements such that $I+E_i$ satisfies T(restriction(p x_i)). As the blank nodes in T(restriction(p x_i)) are disjoint from the blank nodes of T(restriction(p x_j)) for i≠j, $I+E_1+\ldots+E_n$, and thus I, satisfies $T(\text{restriction}(p\ x_1))\cap\ldots\cap T(\text{restriction}(p\ x_n))$. Each restriction(p x_i) is a description, so from the induction hypothesis, M(T(restriction(p x_i)))∈ IOC. The first comprehension principle for sequences then requires that there is some l∈ IL that is a sequence of $I+E_1+\ldots+E_n(M(T(\text{restriction}(p\ x_i))))$,…, $I+E_1+\ldots+E_n(M(T(\text{restriction}(p\ x_i))))$ over IOC. The

comprehension principle for intersectionOf then requires that there is some $y \in IOC$ such that $\langle y,l\rangle \in EXT_I(I(owl:intersectionOf))$ so I satisfies T(d).

For any I+E that satisfies T(d) I+E satisfies $T(d_i)$ so $CEXT_I(I+E(d_i)) = EC(d_i)$. Then $CEXT_I(I+E(M(T(d)))) = CEXT_I(I+E(restriction(p\ x_i)))\cap\ldots\cap CEXT_I(I+E(restriction(p\ x_n))) = EC(restriction(p\ x_i))cup;\ldots\cup EC(restriction(p\ x_i)) = EC(d)$. Finally, $I(M(T(d))) \in IOC$.

Inductive Case: d = restriction(p allValuesFrom(d')), with $p \in VOP \cup VDP$ and d' a description

As $p \in VOP \cup VDP$, from above I satisfies T(p). From the induction hypothesis, I satisfies T(d'). As d' is a description, from the induction hypothesis, any extension of I, I+E, that satisfies T(d') has $I+E(M(T(d'))) = EC(d')$ and $I+E(M(T(d'))) \in IOC$. As $p \in VOP \cup VDP$ and I satisfies T(V'), $I(p) \in IOOP \cup IODP$. A comprehension principle for restriction then requires that I satisfies the triples in T(d) that are not in T(d') or T(p) in a way that shows that I satisfies T(d).

For any I+E that satisfies T(d), $CEXT_I(I+E(M(T(d)))) = \{x \in IOT \mid \forall\ y \in IOT : \langle x,y\rangle \in EXT_I(p) \rightarrow y \in CEXT_I(M(T(d')))\} = \{x \in R \mid \forall\ y \in R : \langle x,y\rangle \in ER(p) \rightarrow y \in EC(d')\} = EC(d)$. Finally, $I+E(M(T(d))) \in IOC$.

Inductive Case: d = restriction(p someValuesFrom(d)) with $p \in VOP \cup VDP$ and d' a description

Similar.

Inductive Case: d = restriction(p allValuesFrom(d)) with $p \in VOP \cup VDP$ and d' a data range

Similar.

Inductive Case: d = restriction(p someValuesFrom(d)) with $p \in VOP \cup VDP$ and d' a data range

Similar.

A.1.2 Lemma 2

Lemma 2: Let V', V, I', and I be as in Lemma 1. Let D be an OWL directive over V'. Then I satisfies T(D) iff I' satisfies D.

Proof

Case: D = Class(foo complete $d_1 \ldots d_n$).

Let d=intersectionOf($d_1 \ldots d_n$). As d is a description over V', thus I satisfies T(d) and for any E mapping the blank nodes of T(d) such that I+E satisfies T(d), $CEXT_I(I+E(M(T(d)))) = EC(d)$. Thus for some E mapping the blank nodes of T(d) such that I+E satisfies T(d), $CEXT_I(I+E(M(T(d)))) = EC(d)$ and $I+E(M(T(d)) \in IOC$.

If I' satisfies D then EC(foo) = EC(d). From above, there is some E such that $CEXT_I(I+E(M(T(d))))$ $= EC(d) = EC(foo) = CEXT_I(I(foo))$ and $I+E(M(T(d)) \in IOC$. Because I satisfies T(V), $I(foo)) \in IOC$, thus $\langle I(foo),I+E(M(T(d)))\rangle \in EXT_I(I(owl:sameClassAs))$. Therefore I satisfies T(D).

If I satisfies T(D) then I satisfies T(intersectionOf($d_1 \ldots d_n$)). Thus there is some E as above such that $\langle I(foo),I+E(M(T(d)))\rangle \in EXT_I(I(owl:sameClassAs))$. Thus $EC(d) = CEXT_I(I+E(M(T(d)))) = CEXT_I(I(foo)) = EC(foo)$. Therefore I' satisfies D.

Case: D = Class(foo partial $d_1 \ldots d_n$)

Let d=intersectionOf($d_1 \ldots d_n$). As d is a description over V', thus I satisfies T(d) and for any E mapping the blank nodes of T(d) such that I+E satisfies T(d), $CEXT_I$(I+E(M(T(d)))) = EC(d). Thus for some E mapping the blank nodes of T(d) such that I+E satisfies T(d), $CEXT_I$(I+E(M(T(d)))) = EC(d) and I+E(M(T(d))$\in$ IOC.

If I' satisfies D then EC(foo) $\subseteq$ EC(d) From above, there is some E such that $CEXT_I$(I+E(M(T(d)))) = EC(d) $\supseteq$ EC(foo) = $CEXT_I$(I(foo)) and I+E(M(T(d))$\in$ IOC. Because I satisfies T(V), I(foo))$\in$ IOC, thus <I(foo),I+E(M(T(d)))> $\in$ EXT_I(I(rdfs:subClassOf)). Therefore I satisfies T(D).

If I satisfies T(D) then I satisfies T(intersectionOf($d_1 \ldots d_n$)). Thus there is some E as above such that <I(foo),I+E(M(T(d)))> $\in$ EXT_I(I(rdfs:subClassOf)). Thus EC(d) = $CEXT_I$(I+E(M(T(d)))) $\supseteq$ $CEXT_I$(I(foo)) = EC(foo). Therefore I' satisfies D.

Case: D = EnumeratedClass(foo $i_1 \ldots i_n$)

Let d=oneOf($i_1 \ldots i_n$). As d is a description over V' so I satisfies T(d) and for some E mapping the blank nodes of T(d) such that I+E satisfies T(d), $CEXT_I$(I+E(M(T(d)))) = EC(d).

If I' satisfies D then EC(foo) = EC(d). From above, there is some E such that $CEXT_I$(I+E(M(T(d)))) = EC(d) = EC(foo) = $CEXT_I$(I(foo)) and I+E(M(T(d))$\in$ IOC. Because I satisfies T(V), I(foo))$\in$ IOC, thus <I(foo),I+E(M(T(d)))> $\in$ EXT_I(I(owl:sameClassAs)). Therefore I satisfies T(D).

If I satisfies T(D) then I satisfies T(intersectionOf($d_1 \ldots d_n$)). Thus there is some E as above such that <I(foo),I+E(M(T(d)))> $\in$ EXT_I(I(owl:sameClassAs)). Thus EC(d) = $CEXT_I$(I+E(M(T(d)))) = $CEXT_I$(I(foo)) = EC(foo). Therefore I' satisfies D.

Case: D= DisjointClasses($d_1 \ldots d_n$)

As d_i is a description over V' therefore I satisfies T(d_i) and for any E mapping the blank nodes of T(d_i) such that I+E satisfies T(d_i), $CEXT_I$(I+E(M(T(d_i)))) = EC(d_i).

If I satisfies T(D) then for each $1 \le i \le n$ there is some E_i such that I satisfies <I+E_i(M(T(d_i))),I+E_j(M(T(d_j)))> $\in$ EXT_I(I(owl:disjointWith)) for each $1 \le i < j \le n$. Thus EC(d_i)$\cap$EC(d_j) = { }, for $i \ne j$. Therefore I' satisfies D.

If I' satisfies D then EC(d_i)$\cup$EC(d_j) = { } for $i \ne j$. For any E_i and E_j as above <I+E_i+E_j(M(T(d_i))),I+E_i+E_j(M(T(d_j)))< $\in$ EXT_I(I(owl:disjointWith)), for $i \ne j$. As at least one E_i exists for each i, and the blank nodes of the T(d_j) are all disjoint, I+E_1+…+E_n satisfies T(DisjointClasses($d_1 \ldots d_n$)). Therefore I satisfies T(D).

Case: D = EquivalentClasses(d_1 … d_n)

Similar.

Case: D = SubClassOf(d_1 d2)

Somewhat similar.

Case: D = IndividualProperty(p super(s_1) … super(sn) domain(d_1) … domain(d_m) range(r_1) … range(r_1) [inverse(i)] [Symmetric] [Functional] [InverseFunctional] [OneToOne] [Transitive])

As d_i for $1 \le i \le m$ is a description over V' therefore I satisfies $T(d_i)$ and for any E mapping the blank nodes of $T(d_i)$ such that I+E satisfies $T(d_i)$, $CEXT_I(I+E(M(T(d_i)))) = EC(d_i)$. Similarly for r_i for $1 \le i \le l$.

If I' satisfies D, then, as $p \in VOP$, I satisfies $I(p) \in IOOP$. Then, as I is an OWL DL interpretation, I satisfies $\langle I(p),I(owl:Thing)\rangle \in EXT_I(I(rdfs:domain))$ and $\langle I(p),I(owl:Thing)\rangle \in EXT_I(I(rdfs:range))$. Also, $ER(p) \subseteq ER(s_i)$ for $1 \le i \le n$, so $EXT_I(I(p))=ER(p) \subseteq ER(s_i)=EXT_I(I(s_i))$ and I satisfies $\langle I(p),I(s_i)\rangle \in EXT_I(I(rdfs:subPropertyOf))$. Next, $ER(p) \subseteq EC(d_i) \times R$ for $1 \le i \le m$, so $\langle z,w\rangle \in ER(p) \rightarrow z \in EC(d_i)$ and for any E such that I+E satisfies $T(d_i)$, $\langle z,w\rangle \in EXT_I(p) \rightarrow z \in CEXT_I(I+E(M(T(d_i))))$ and thus $\langle I(p),I+E(M(T(d_i)))\rangle \in EXT_I(I(rdfs:domain))$. Similarly for r_i) for $1 \le i \le l$.

If I' satisfies D and inverse(i) is in D, then ER(p) and ER(i) are converses. Thus $\langle u,v\rangle \in ER(p)$ iff $\langle v,u\rangle \in ER(i)$ so $\langle u,v\rangle \in EXT_I(p)$ iff $\langle v,u\rangle \in EXT_I(i)$ and I satisfies $\langle I(p),I(i)\rangle \in EXT_I(I(owl:inverseOf))$. If I' satisfies D and Symmetric is in D, then ER(p) is symmetric. Thus if $\langle x,y\rangle \in ER(p)$ then $\langle y,x\rangle \in ER(p)$ so if $\langle x,y\rangle \in EXT_I(p)$ then $\langle y, x\rangle \in EXT_I(p)$. and thus I satisfies $p \in CEXT_I(I(owl:Symmetric))$. Similarly for Functional, InverseFunctional, and Transitive. Thus if I' satisfies D then I satisfies T(D).

If I satisfies T(D) then, for $1 \le i \le n$, $\langle I(p),I(s_i)\rangle \in EXT_I(I(rdfs:subPropertyOf))$. so $ER(p)=EXT_I(I(p)) \subseteq EXT_I(I(s_i))=ER(s_i)$. Also, for $1 \le i \le m$, for some E such that I+E satisfies $T(d_i)$, $\langle I(p),I+E(M(T(d_i)))\rangle \in EXT_I(I(rdfs:domain))$ so $\langle z,w\rangle \in EXT_I(p) \rightarrow z \in CEXT_I(I+E(M(T(d_i))))$. Thus $\langle z,w\rangle \in ER(p) \rightarrow z \in EC(d_i)$ and $ER(p) \subseteq EC(d_i) \times R$. Similarly for r_i) for $1 \le i \le l$.

If I satisfies T(D) and inverse(i) is in D, then I satisfies $\langle I(p),I(i)\rangle \in EXT_I(I(owl:inverseOf))$. Thus $\langle u,v\rangle \in EXT_I(p)$ iff $\langle v,u\rangle \in EXT_I(i)$ so $\langle u,v\rangle \in ER(p)$ iff $\langle v,u\rangle \in ER(i)$ and ER(p) and ER(i) are converses. If I satisfies D and Symmetric is in D, then I satisfies $p \in CEXT_I(I(owl:Symmetric))$ so if $\langle x,y\rangle \in EXT_I(p)$ then $\langle y, x\rangle \in EXT_I(p)$. Thus if $\langle x,y\rangle \in ER(p)$ then $\langle y,x\rangle \in ER(p)$ and ER(p) is symmetric. Similarly for Functional, InverseFunctional, and Transitive. Thus if I satisfies T(D) then I' satisfies D.

Case: D = DataProperty(p super(s_1) … super(sn) domain(d_1) … domain(d_m) range(r_1) … range(r_1) [Functional])

Similar.

Case: D = EquivalentProperties(p_1 … p_n), for $p_i \in VOP$

As $p_i \in$ VOP and I satisfies T(V'), $I(p_i) \in$ IOP. If I satisfies T(D) then $\langle I(p_i), I(p_j) \rangle \in EXT_I(I(\text{owl:samePropertyAs}))$, for each $1 \le i < j \le n$. Therefore $EXT_I(p_i) = EXT_I(p_j)$, for each $1 \le i < j \le n$; $CR(p_i) = CR(p_j)$, for each $1 \le i < j \le n$; and I' satisfies D.

If I' satisfies D then $CR(p_i) = CR(p_j)$, for each $1 \le i < j \le n$. Therefore $EXT_I(p_i) = EXT_I(p_j)$, for each $1 \le i < j \le n$. From the OWL DL definition of owl:samePropertyAs, $\langle I(p_i), I(p_j) \rangle \in EXT_I(I(\text{owl:samePropertyAs}))$, for each $1 \le i < j \le n$. Thus I satisfies T(D).

Case: D = SubPropertyOf(p_1 p_2)

Somewhat similar, but simpler.

Case: D = SameIndividual(i_1 … i_n)

Similar to SamePropertyAs.

Case: D = DifferentIndividuals(i_1 … i_n)

Similar to SamePropertyAs.

Case: D = Individual(i type(t_1) … type(t_n) value(p_1 v_1) … value(p_n v_n))

If I satisfies T(D) then there is some E that maps each blank node in T(D) such that I+E satisfies T(D). A simple examination of T(D) shows that the mappings of E plus the mappings for the individual IDs in D, which are all in IOT, show that I' satisfies D.

If I' satisfies D then for each Individual construct in D there must be some element of R that makes the type relationships and relationships true in D. The triples in T(D) then fall into three categories. 1/ Type relationships to owl:Thing, which are true in I because the elements above belong to R. 2/ Type relationships to OWL descriptions, which are true in I because they are true in I', from Lemma 1. 3/ OWL property relationships, which are true in I' because they are true in I. Thus I satisfies T(D).

A.1.3 Lemma 3

Lemma 3: Let V' = <VI, VC, VD, VOP, VDP> be a separated OWL vocabulary. Let V = VI ∪ VC ∪ VD ∪ VOP ∪ VDP ∪ VRDFS ∪ VOWL. Then for every OWL DL interpretation $I = \langle R_I, EXT_I, S_I, L_I \rangle$ over V that satisfies T(V') there is an abstract OWL interpretation I' over V' such that for any OWL abstract KB K over V, I' abstract OWL satisfies K iff I OWL DL satisfies T(K).

Proof

1. Let $CEXT_I$ be defined as usual from I. The required abstract OWL interpretation will be $I' = \langle CEXT_I(I(\text{owl:Thing})), EC, ER, S \rangle$ where S(n) = I(n) for $n \in$ VI, $EC(n) = CEXT_I(I(n))$ for $n \in VC \cup VD$, and $ER(n) = EXT_I(I(n))$ for $n \in VOP \cup VDP$.

2. V', V, I', and I meet the requirements of Lemma 2, so for any directive D over V' I satisfies T(D) iff I' satisfies D.

3. Satisfying an abstract KB is just satisfying its directives and satisfying the translation of an abstract KB is just satisfying all the triples so I OWL DL satisfies T(K) iff I' abstract OWL satisfies K.

A.1.4 Lemma 4

Lemma 4: Let V' = <VI, VC, VD, VOP, VDP> be a separated OWL vocabulary. Let V = VI ∪ VC ∪ VD ∪ VOP ∪ VDP ∪ VRDFS ∪ VOWL. Then for every Abstract OWL interpretation I' = < U, EC, ER, S > over V' there is an OWL DL interpretation I over V that satisfies T(V') such that for any abstract OWL KB K over V', I OWL DL satisfies T(K) iff I' abstract OWL satisfies K.

Proof

1. Construct I = < R_I, EXT_I, S_I, L_I > as follows:
   - Let IOC be the OWL descriptions over V', i.e., C is part of the Herbrand universe of OWL syntax.
     Let OD be the OWL data ranges over V'.
     Let OP = VOP u VDP.
     Let OL be the finite sequences over U, over C, and over LV plus rdf:nil.
     Let OX = VRDFS∪VOWL - {rdf:nil}.
   - L_I be any mapping compatible with D.
   - R_I = U ∪ IOC ∪ OD ∪ OP ∪ OL ∪ OX
   - Let I(n) = S(n) for n∈ VI.
     Let I(n) = n for n∈ V-VI.
   - For i∈ U, EXT_I(i)={ } and $CEXT_I$(i)={ }.
     For c∈ IOC, EXT_I(c)={ } and $CEXT_I$(c)=EC(c).
     For d∈ OD, EXT_I(d)={ } and $CEXT_I$(d)=EC(d).
     For r∈ OP, EXT_I(r)=ER(r) and $CEXT_I$(r)={ }.
     For l∈ OL, EXT_I(r)={ } and $CEXT_I$(r)={ }.
   - $CEXT_I$(rdf:first)={ }, $CEXT_I$(rdf:rest)={ }
     $CEXT_I$(rdf:nil)={ }, EXT_I(rdf:nil)={ }
     $CEXT_I$(rdf:List)=OL, EXT_I(rdf:List)={ }
     <x,y> ∈ EXT_I(rdf:first) iff x∈ OL and y is its first element
     <x,y> ∈ EXT_I(rdf:rest) iff x∈ OL and y is its tail
   - owl:intersectionOf, owl:oneOf, owl:unionOf, owl:complementOf, owl:onProperty, owl:allValuesFrom, owl:someValuesFrom, owl:hasValue, owl:minCardinality, owl:maxCardinality, and owl:cardinality have empty class extensions and their extensions are as necessary to link the elements of IOC up with their parts.
2. $CEXT_I$(rdfs:range) = { };
   EXT_I(rdfs:range) = { <x,y> : x∈ OP y∈ IOC ∧ ∀ z, <w,z> ∈ EXT_I(x) → z ∈ $CEXT_I$(y); } ∪ [RDFS ranges]
   $CEXT_I$(rdfs:subClassOf) = { };
   $CEXT_I$(rdfs:Datatype) = D;
   EXT_I(rdfs:subClassOf) = { <x,y> : x,y∈ IOC ∧ $CEXT_I$(x) ⊆ $CEXT_I$(y) } ∪ [RDFS subclasses]
   $CEXT_I$(rdfs:subPropertyOf) = { };
   EXT_I(rdfs:subPropertyOf) = { <x,y> : x,y∈ OP ∧ EXT_I(x) ⊆ EXT_I(y) } ∪ [RDFS subproperties]
   $CEXT_I$(rdf:type) = { }, EXT_I(rdf:type) is determined by $CEXT_I$. Then I is an OWL DL

interpretation because the conditions for the class extensions in OWL DL match up with the conditions for class-like OWL abstract syntax constructs.

3. V', V, I', and I meet the requirements of Lemma 2, so for any directive D over V' I satisfies T(D) iff I' satisfies D.
4. Satisfying an abstract KB is just satisfying its directives and satisfying the translation of an abstract KB is just satisfying all the triples so I OWL DL satisfies T(K) iff I' abstract OWL satisfies K.

A.1.5 Correspondence Theorem

Theorem 1: Let V' be a separated OWL vocabulary. Let K,Q be abstract OWL ontologies with separated names over V'. Then K OWL entails Q iff T(K),T(V') OWL DL entails T(Q).

Proof

Suppose K OWL entails Q. Let I be an OWL DL interpretation that satisfies T(K),T(V'). Then from Lemma 3, there is some abstract OWL interpretation I' such that for any abstract OWL ontology X over V', I satisfies T(X) iff I' satisfies X. Thus I' satisfies K. Because K OWL entails Q, I' satisfies Q, so I satisfies T(Q). Thus T(K),T(V') OWL DL entails T(Q).

Suppose T(K),T(V') OWL DL entails T(Q). Let I' be an abstract OWL interpretation that satisfies K. Then from Lemma 4, there is some OWL DL interpretation I that satisfies T(V') such that for any abstract OWL ontology X over V', I satisfies T(X) iff I' satisfies X. Thus I satisfies T(K). Because T(K),T(V') OWL DL entails T(Q), I satisfies T(Q), so I' satisfies Q. Thus K abstract OWL entails Q.

A.2 Correspondence between OWL DL and OWL Full

This section contains a proof sketch concerning the relationship between OWL DL and OWL Full. This proof has not been fully worked out. Significant effort may be required to finish the proof and some details of the relationship may have to change.

Let K be an RDF Graph. An OWL interpretation of of K is an OWL interpretation (from Section 5.2) that is an RDFS interpretation of K.

Lemma 5: Let V be a separated vocabulary. Then for every OWL intepretation I there is an OWL DL interpretation I' (as in Section 5.3) such that for K any OWL KB in the abstract syntax with separated vocabulary V, I is an OWL interpretation of T(K) iff I' is an OWL DL interpretation of T(K).

Proof sketch: As all OWL DL interpretations are OWL interpretations, the reverse direction is obvious.

Let $I = < R_I, EXT_I, S_I, L_I >$ be an OWL interpretation of T(K). Let $I' = < R_{I'}, EXT_{I'}, S_{I'}, L_{I'} >$ be an OWL interpretation of T(K). Let $R_{I'} = CEXT_I(I(owl:Thing)) + CEXT_I(I(owl:ObjectProperty)) + CEXT_I(I(owl:IndividualProperty)) + CEXT_I(I(owl:Class)) + CEXT_I(I(rdf:List)) + R_I$, where + is disjoint union. Define $EXT_{I'}$ so as to separate the various roles of the copies. Define $S_{I'}$ so as to map vocabulary into the appropriate copy. This works because K has a separated vocabulary, so I can be split according the the roles, and there are no inappropriate relationships in EXT_I. In essence the first component of $R_{I'}$ is OWL individuals, the second component of $R_{I'}$ is OWL datatype properties, the third

component of $R_{I'}$ is OWL object properties, the fourth component of $R_{I'}$ is OWL classes, the fifth component of $R_{I'}$ is RDF lists, and the sixth component of $R_{I'}$ is everything else.

Theorem 2: Let K,C be RDF graphs such that each of K, C, and K∪C is the translation of some OWL KB in the abstract syntax with separated vocabulary. Then K OWL Full entails C if K OWL DL entails C.

Proof: From the above lemma and because all OWL Full interpretations are OWL interpretations.

Comment: The only if direction cannot be proved without showing that OWL Full has no semantic oddities, which has not yet been done.

Appendix B. Examples (Informative)

This appendix gives examples of the concepts developed in the rest of the document.

B.1 Examples of Mapping from Abstract Syntax to RDF Graphs

The transformation rules in Section 4 transform

```
DatatypeProperty(ex:name)
ObjectProperty(ex:author)
Individual(type(ex:Book)
      value(ex:author Individual(type(ex:Person) value(ex:name xsd:string"Fred"))))
```

to

```
ex:name rdf:type owl:DatatypeProperty .
ex:author rdf:type owl:ObjectProperty .
ex:Book rdf:type owl:Class .
ex:Person rdf:type owl:Class .

_:x rdf:type ex:Book .
_:x ex:author _:x1 .
_:x1 rdf:type ex:Person .
_:x1 ex:name "Fred"^^xsd:string .
```

and

```
ObjectProperty(ex:enrolledIn)
Class(ex:Student complete ex:Person
                restriction(ex:enrolledIn allValuesFrom(ex:School) minCardinality(1)))
```

to

```
ex:enrolledIn rdf:type owl:ObjectProperty .

ex:Person rdf:type owl:Class .
ex:School rdf:type owl:Class .

ex:Student rdf:type owl:Class .
ex:Student owl:sameClassAs _:x .
```

```
_:x owl:intersectionOf _:l1 .
_:l1 rdf:type rdf:List .
_:l1 rdf:first  ex:Person .
_:l1 rdf:rest _:l2 .
_:l2 rdf:type rdf:List .
_:l2 rdf:first  _:lr .
_:l2 rdf:rest rdf:nil .

_:lr owl:intersectionOf _:lr1 .
_:lr1 rdf:type rdf:List .
_:lr1 rdf:first _:r1 .
_:lr1 rdf:rest _:lr2 .
_:lr2 rdf:type rdf:List .
_:lr2 rdf:first _:r2 .
_:lr2 rdf:rest rdf:nil .

_:r1 rdf:type owl:Restriction .
_:r1 owl:onProperty ex:enrolledIn .
_:r1 owl:allValuesFrom ex:School .

_:r2 rdf:type owl:Restriction .
_:r2 owl:onProperty ex:enrolledIn .
_:r2 owl:minCardinality "1"^^xsd:nonNegativeInteger .
```

B.2 Examples of Entailments in OWL DL and OWL Full

OWL DL supports the entailments that one would expect, as long as the vocabulary can be shown to belong to the appropriate piece of the domain of discourse. For example,

```
John friend Susan .
```

does not OWL DL entail

```
John rdf:type owl:Thing .
Susan rdf:type owl:Thing .
friend rdf:type owl:ObjectProperty .
```

The above three triples would have to be added before the following restriction could be concluded

```
John rdf:type _:x .
_:x owl:onProperty friend .
_:x owl:minCardinality 1 .
```

However, once this extra information is added, all natural entailments follow, except for those that involve descriptions with loops. For example,

```
John rdf:type owl:Thing .
friend rdf:type owl:ObjectProperty .
John rdf:type _:x .
_:x owl:onProperty friend .
_:x owl:maxCardinality 0 .
```

does not entail

```
John rdf:type _:y .
_:y owl:onProperty friend .
_:y owl:allValuesFrom _:y .
```

because there are no comprehension principles for such looping descriptions. It is precisely the lack of such comprehension principles that prevent the formation of paradoxes in OWL DL while still retaining natural entailments.

In OWL DL one can repair missing localizations in any separated-syntax KB by adding a particular set of localizing assertions consisting of all triples of the form

```
<individual> rdf:type owl:Thing .
<class> rdf:type owl:Class .
<oproperty> rdf:type owl:ObjectProperty .
<dtproperty> rdf:type owl:DatatypeProperty .
```

Call the result of adding all such assertions to a OWL DL KB the *localization* of the KB.

OWL Full supports the entailments that one would expect, and there is no need to provide typing information for the vocabulary. For example,

```
John friend Susan .
```

does OWL Full entail

```
John rdf:type _:x .
_:x owl:onProperty friend .
_:x owl:minCardinality 1 .
```