OWL 2 Web Ontology Language Profiles

W3C Editor's Draft 21 May 2009

This version:
http://www.w3.org/2007/OWL/draft/ED-owl2-profiles-20090521/
Latest editor's draft:
http://www.w3.org/2007/OWL/draft/owl2-profiles/
Previous version:
http://www.w3.org/2007/OWL/draft/ED-owl2-profiles-20090421/ (color-coded diff)

Editors:
Boris Motik, Oxford University
Bernardo Cuenca Grau, Oxford University
Ian Horrocks, Oxford University
Zhe Wu, Oracle
Achille Fokoue, IBM
Carsten Lutz, University of Bremen
Contributors: (in alphabetical order)
Diego Calvanese, Free University of Bozen-Bolzano
Jeremy Carroll, TopQuadrant
Giuseppe De Giacomo, Sapienza Università di Roma
Ivan Herman, W3C/ERCIM
Bijan Parsia, University of Manchester
Peter F Patel-Schneider, Bell Labs Research, Alcatel-Lucent
Alan Ruttenberg, Science Commons (Creative Commons)
Uli Sattler, University of Manchester

This document is also available in these non-normative formats: PDF version.

Abstract

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies provide classes, properties, individuals, and data values and are stored as Semantic Web

documents. OWL 2 ontologies can be used along with information written in RDF, and OWL 2 ontologies themselves are primarily exchanged as RDF documents. The OWL 2 Document Overview describes the overall state of OWL 2, and should be read before other OWL 2 documents.
This document provides a specification of several profiles of OWL 2 which can be more simply and/or efficiently implemented. In logic, profiles are often called fragments. Most profiles are defined by placing restrictions on the structure of OWL 2 ontologies. These restrictions have been specified by modifying the productions of the functional-style syntax.

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Summary of Changes

This Last Call Working Draft has undergone only minor changes since the previous version of 21st April, 2009.

- One grammar production was fixed in order to align it with the grammar in Structural Specification and Functional-Style Syntax
- A reference to logic programming has been added.
- Some minor editorial changes were made.

Please Comment By 9 July 2009

The OWL Working Group seeks public feedback on this Working Draft. Please send your comments to public-owl-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document and see if the relevant text has already been updated.

No Endorsement

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

Table of Contents

1 Introduction

An OWL 2 *profile* (commonly called a *fragment* or a *sublanguage* in computational logic) is a trimmed down version of OWL 2 that trades some expressive power for the efficiency of reasoning. This document describes three profiles of OWL 2, each of which achieves efficiency in a different way and is useful in different application scenarios. The profiles are independent of each other, so (prospective) users can skip over the descriptions of profiles that are not of interest to them. The choice of which profile to use in practice will depend on the structure of the ontologies and the reasoning tasks at hand (see Section 10 of the OWL 2 Primer [*OWL 2 Primer*] for more help in understanding and selecting profiles).

- **OWL 2 EL** is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes. This profile captures the expressive power used by many such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology [*EL++*] (see Section 5 for more information on computational complexity). Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way.
- **OWL 2 QL** is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. In OWL 2 QL, conjunctive query answering can be implemented using conventional relational database systems. Using a suitable reasoning technique, sound and complete conjunctive query answering can be performed in LOGSPACE with respect to the size of the data (assertions). As in OWL 2 EL, polynomial time algorithms can be used to implement the ontology consistency and class expression subsumption reasoning problems. The expressive power of the profile is necessarily quite limited, although it does include most of the main features of conceptual models such as UML class diagrams and ER diagrams.
- **OWL 2 RL** is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity. OWL 2 RL reasoning systems can be implemented using rule-based reasoning engines. The ontology consistency, class expression satisfiability, class expression subsumption, instance checking, and conjunctive query answering problems can be solved in time that is polynomial with respect to the size of the ontology.

OWL 2 profiles are defined by placing restrictions on the structure of OWL 2 ontologies. Syntactic restrictions can be specified by modifying the grammar of the functional-style syntax [*OWL 2 Specification*] and possibly giving additional global restrictions. In this document, the modified grammars are specified in two ways. In each profile definition, only the difference with respect to the full grammar is given; that is, only the productions that differ from the functional-style syntax are presented, while the productions that are the same as in the functional-style syntax are not repeated. Furthermore, the full grammar for each of the profiles is given in the Appendix.

An ontology in any profile can be written into an ontology document by using any of the syntaxes of OWL 2.

Apart from the ones specified here, there are many other possible profiles of OWL 2 — there are, for example, a whole family of profiles that extend OWL 2 QL. This document does not list OWL Lite [*OWL 1 Reference*]; however, all OWL Lite ontologies are OWL 2 ontologies, so OWL Lite can be viewed as a profile of OWL 2. Similarly, OWL 1 DL can also be viewed as a profile of OWL 2.

The italicized keywords *must*, *must not*, *should*, *should not*, and *may* are used to specify normative features of OWL 2 documents and tools, and are interpreted as specified in RFC 2119 [*RFC 2119*].

Features At Risk

This document depends on the two features identified in the OWL 2 Specification [*OWL 2 Specification*] as being at risk: support for *owl:rational*, and support for *rdf:XMLLiteral*. If those features are removed from OWL 2, future versions of this document will be updated accordingly.

Please send feedback to public-owl-comments@w3.org. For current status see features "at risk" in OWL 2.

2 OWL 2 EL

The OWL 2 EL profile [*EL++*,*EL++ Update*] is designed as a subset of OWL 2 that

- is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties,
- captures the expressive power used by many such ontologies, and
- for which ontology consistency, class expression subsumption, and instance checking can be decided in polynomial time.

For example, OWL 2 EL provides class constructors that are sufficient to express the very large biomedical ontology SNOMED CT [*SNOMED CT*].

2.1 Feature Overview

OWL 2 EL places restrictions on the type of class restrictions that can be used in axioms. In particular, the following types of class restrictions are supported:

- existential quantification to a class expression (**ObjectSomeValuesFrom**) or a data range (**DataSomeValuesFrom**)
- existential quantification to an individual (**ObjectHasValue**) or a literal (**DataHasValue**)
- self-restriction (**ObjectHasSelf**)
- enumerations involving a *single* individual (**ObjectOneOf**) or a *single* literal (**DataOneOf**)
- intersection of classes (**ObjectIntersectionOf**) and data ranges (**DataIntersectionOf**)

OWL 2 EL supports the following axioms, all of which are restricted to the allowed set of class expressions:

- class inclusion (**SubClassOf**)
- class equivalence (**EquivalentClasses**)
- class disjointness (**DisjointClasses**)
- object property inclusion (**SubObjectPropertyOf**) with or without property chains, and data property inclusion (**SubDataPropertyOf**)
- property equivalence (**EquivalentObjectProperties** and **EquivalentDataProperties**),
- transitive object properties (**TransitiveObjectProperty**)
- reflexive object properties (**ReflexiveObjectProperty**)
- domain restrictions (**ObjectPropertyDomain** and **DataPropertyDomain**)
- range restrictions (**ObjectPropertyRange** and **DataPropertyRange**)
- assertions (**SameIndividual**, **DifferentIndividuals**, **ClassAssertion**, **ObjectPropertyAssertion**, **DataPropertyAssertion**, **NegativeObjectPropertyAssertion**, and **NegativeDataPropertyAssertion**)
- functional data properties (**FunctionalDataProperty**)
- keys (**HasKey**)

The following constructs are not supported in OWL 2 EL:

- universal quantification to a class expression (**ObjectAllValuesFrom**) or a data range (**DataAllValuesFrom**)
- cardinality restrictions (**ObjectMaxCardinality**, **ObjectMinCardinality**, **ObjectExactCardinality**, **DataMaxCardinality**, **DataMinCardinality**, and **DataExactCardinality**)
- disjunction (**ObjectUnionOf**, **DisjointUnion**, and **DataUnionOf**)
- class negation (**ObjectComplementOf**)
- enumerations involving more than one individual (**ObjectOneOf** and **DataOneOf**)
- disjoint properties (**DisjointObjectProperties** and **DisjointDataProperties**)
- irreflexive object properties (**IrreflexiveObjectProperty**)
- inverse object properties (**InverseObjectProperties**)

W3C Editor's Draft

- functional and inverse-functional object properties (**FunctionalObjectProperty** and **InverseFunctionalObjectProperty**)
- symmetric object properties (**SymmetricObjectProperty**)
- asymmetric object properties (**AsymmetricObjectProperty**)

2.2 Profile Specification

The following sections specify the structure of OWL 2 EL ontologies.

2.2.1 Entities

Entities are defined in OWL 2 EL in the same way as in the structural specification [*OWL 2 Specification*], and OWL 2 EL supports all predefined classes and properties. Furthermore, OWL 2 EL supports the following datatypes:

- *rdf:text*
- *rdf:XMLLiteral*
- *rdfs:Literal*
- *owl:real*
- *owl:rational*
- *xsd:decimal*
- *xsd:integer*
- *xsd:nonNegativeInteger*
- *xsd:string*
- *xsd:normalizedString*
- *xsd:token*
- *xsd:Name*
- *xsd:NCName*
- *xsd:NMTOKEN*
- *xsd:hexBinary*
- *xsd:base64Binary*
- *xsd:anyURI*
- *xsd:dateTime*
- *xsd:dateTimeStamp*

The set of supported datatypes has been designed such that the intersection of the value spaces of any set of these datatypes is either empty or infinite, which is necessary to obtain the desired computational properties [*EL++*]. Consequently, the following datatypes *must not* be used in OWL 2 EL: *xsd:double*, *xsd:float*, *xsd:nonPositiveInteger*, *xsd:positiveInteger*, *xsd:negativeInteger*, *xsd:long*, *xsd:int*, *xsd:short*, *xsd:byte*, *xsd:unsignedLong*, *xsd:unsignedInt*, *xsd:unsignedShort*, *xsd:unsignedByte*, *xsd:language*, and *xsd:boolean*.

Finally, OWL 2 EL does not support anonymous individuals.

```
Individual := NamedIndividual
```

2.2.2 Property Expressions

Inverse properties are not supported in OWL 2 EL, so object property expressions are restricted to named properties. Data property expressions are defined in the same way as in the structural specification [OWL 2 Specification].

```
ObjectPropertyExpression := ObjectProperty
```

2.2.3 Class Expressions

In order to allow for efficient reasoning, OWL 2 EL restricts the set of supported class expressions to **ObjectIntersectionOf**, **ObjectSomeValuesFrom**, **ObjectHasSelf**, **ObjectHasValue**, **DataSomeValuesFrom**, **DataHasValue**, and **ObjectOneOf** containing a single individual.

```
ClassExpression :=
    Class | ObjectIntersectionOf | ObjectOneOf |
    ObjectSomeValuesFrom | ObjectHasValue | ObjectHasSelf |
    DataSomeValuesFrom | DataHasValue
ObjectOneOf := 'ObjectOneOf' '(' Individual ')'
```

2.2.4 Data Ranges

A data range expression is restricted in OWL 2 EL to the predefined datatypes admitted in OWL 2 EL, intersections of data ranges, and to enumerations of literals consisting of a single literal.

```
DataRange := Datatype | DataIntersectionOf | DataOneOf
DataOneOf := 'DataOneOf' '(' Literal ')'
```

2.2.5 Axioms

The class axioms of OWL 2 EL are the same as in the structural specification [OWL 2 Specification], with the exception that **DisjointUnion** is disallowed. Different class axioms are defined in the same way as in the structural specification [OWL 2 Specification], with the difference that they use the new definition of **ClassExpression**.

ClassAxiom := **SubClassOf** | **EquivalentClasses** | **DisjointClasses**

OWL 2 EL supports the following object property axioms, which are defined in the same way as in the structural specification [OWL 2 Specification], with the difference that they use the new definition of **ObjectPropertyExpression**.

ObjectPropertyAxiom :=
  EquivalentObjectProperties | **SubObjectPropertyOf** |
  ObjectPropertyDomain | **ObjectPropertyRange** |
  ReflexiveObjectProperty | **TransitiveObjectProperty**

OWL 2 EL provides the same axioms about data properties as the structural specification [OWL 2 Specification] apart from **DisjointDataProperties**.

DataPropertyAxiom :=
  SubDataPropertyOf | **EquivalentDataProperties** |
  DataPropertyDomain | **DataPropertyRange** | **FunctionalDataProperty**

The assertions in OWL 2 EL, as well as all other axioms, are the same as in the structural specification [OWL 2 Specification], with the difference that class object property expressions are restricted as defined in the previous sections.

2.2.6 Global Restrictions

OWL 2 EL extends the global restrictions on axioms from Section 11 of the structural specification [OWL 2 Specification] with an additional condition [EL++ Update]. In order to define this condition, the following notion is used.

The set of axioms `Ax` *imposes a range restriction to a class expression* `CE` *on an object property* OP_1 if `Ax` contains the following axioms, where k ≥ 1 is an integer and OP_i are object properties:

$$\mathtt{SubObjectPropertyOf(}\ OP_1\ OP_2\mathtt{)}$$
$$\ldots$$
$$\mathtt{SubObjectPropertyOf(}\ OP_{k-1}\ OP_k\ \mathtt{)}$$
$$\mathtt{ObjectPropertyRange(}\ OP_k\ \mathtt{CE}\ \mathtt{)}$$

The axiom closure `Ax` of an OWL 2 EL ontology *must* obey the restrictions described in Section 11 of the structural specification [OWL 2 Specification] and, in addition, if

- `Ax` contains `SubObjectPropertyOf( ObjectPropertyChain(` OP_1 `...` OP_n `) OP )` and
- `Ax` imposes a range restriction to some class expression `CE` on `OP`

then `Ax` *must* impose a range restriction to `CE` on OP_n.

This additional restriction is vacuously true for each **SubObjectPropertyOf** axiom in which in the first item of the previous definition does not contain a property chain. There are no additional restrictions for range restrictions on reflexive and transitive roles — that is, a range restriction can be placed on a reflexive and/or transitive role provided that it satisfies the previously mentioned restriction.

3 OWL 2 QL

The OWL 2 QL profile is designed so that sound and complete query answering is in LOGSPACE (more precisely, in AC^0) with respect to the size of the data (assertions), while providing many of the main features necessary to express conceptual models such as UML class diagrams and ER diagrams. In particular, this profile contains the intersection of RDFS and OWL 2 DL. It is designed so that data (assertions) that is stored in a standard relational database system can be queried through an ontology via a simple rewriting mechanism, i.e., by rewriting the query into an SQL query that is then answered by the RDBMS system, without any changes to the data.

OWL 2 QL is based on the DL-Lite family of description logics [*DL-Lite*]. Several variants of DL-Lite have been described in the literature, and DL-Lite$_R$ provides the logical underpinning for OWL 2 QL. DL-Lite$_R$ does not require the unique name assumption (UNA), since making this assumption would have no impact on the semantic consequences of a DL-Lite$_R$ ontology. More expressive variants of DL-Lite, such as DL-Lite$_A$, extend DL-Lite$_R$ with functional properties, and these can also be extended with keys; however, for query answering to remain in LOGSPACE, these extensions require UNA and need to impose certain global restrictions on the interaction between properties used in different types of axiom. Basing OWL 2 QL on DL-Lite$_R$ avoids practical problems involved in the explicit axiomatization of UNA. Other variants of DL-Lite can also be supported on top of OWL 2 QL, but may require additional restrictions on the structure of ontologies.

3.1 Feature Overview

OWL 2 QL is defined not only in terms of the set of supported constructs, but it also restricts the places in which these constructs are allowed to occur. The allowed usage of constructs in class expressions is summarized in Table 1.

Table 1. Syntactic Restrictions on Class Expressions in OWL 2 QL

Subclass Expressions	Superclass Expressions
a class existential quantification	a class intersection (**ObjectIntersectionOf**)

(**ObjectSomeValuesFrom**) where the class is limited to *owl:Thing* existential quantification to a data range (**DataSomeValuesFrom**)	negation (**ObjectComplementOf**) existential quantification to a class (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**)

OWL 2 QL supports the following axioms, constrained so as to be compliant with the mentioned restrictions on class expressions:

- subclass axioms (**SubClassOf**)
- class expression equivalence (**EquivalentClasses**)
- class expression disjointness (**DisjointClasses**)
- inverse object properties (**InverseObjectProperties**)
- property inclusion (**SubObjectPropertyOf** not involving property chains and **SubDataPropertyOf**)
- property equivalence (**EquivalentObjectProperties** and **EquivalentDataProperties**)
- property domain (**ObjectPropertyDomain** and **DataPropertyDomain**)
- property range (**ObjectPropertyRange** and **DataPropertyRange**)
- disjoint properties (**DisjointObjectProperties** and **DisjointDataProperties**)
- symmetric properties (**SymmetricObjectProperty**)
- reflexive properties (**ReflexiveObjectProperty**)
- irreflexive properties (**IrreflexiveObjectProperty**)
- asymmetric properties (**AsymmetricObjectProperty**)
- assertions other than individual equality assertions and negative property assertions (**DifferentIndividuals**, **ClassAssertion**, **ObjectPropertyAssertion**, and **DataPropertyAssertion**)

The following constructs are not supported in OWL 2 QL:

- existential quantification to a class expression or a data range (**ObjectSomeValuesFrom** and **DataSomeValuesFrom**) in the subclass position
- self-restriction (**ObjectHasSelf**)
- existential quantification to an individual or a literal (**ObjectHasValue**, **DataHasValue**)
- enumeration of individuals and literals (**ObjectOneOf**, **DataOneOf**)
- universal quantification to a class expression or a data range (**ObjectAllValuesFrom**, **DataAllValuesFrom**)
- cardinality restrictions (**ObjectMaxCardinality**, **ObjectMinCardinality**, **ObjectExactCardinality**, **DataMaxCardinality**, **DataMinCardinality**, **DataExactCardinality**)
- disjunction (**ObjectUnionOf**, **DisjointUnion**, and **DataUnionOf**)
- property inclusions (**SubObjectPropertyOf**) involving property chains
- functional and inverse-functional properties (**FunctionalObjectProperty**, **InverseFunctionalObjectProperty**, and **FunctionalDataProperty**)
- transitive properties (**TransitiveObjectProperty**)
- keys (**HasKey**)
- individual equality assertions and negative property assertions

OWL 2 QL does not support individual equality assertions (**SameIndividual**): adding such axioms to OWL 2 QL would increase the data complexity of query answering, so that it is no longer first order rewritable, which means that query answering could not be implemented directly using relational database technologies. However, an ontology *O* that includes individual equality assertions, but is otherwise OWL 2 QL, could be handled by computing the reflexive–symmetric–transitive closure of the equality (**SameIndividual**) relation in *O* (this requires answering recursive queries and can be implemented in LOGSPACE w.r.t. the size of data) [*DL-Lite-bool*], and then using this relation in query answering procedures to simulate individual equality reasoning [*Automated Reasoning*].

3.2 Profile Specification

The productions for OWL 2 QL are defined in the following sections. Note that each OWL 2 QL ontology must satisfy the global restrictions on axioms defined in Section 11 of the structural specification [*OWL 2 Specification*].

3.2.1 Entities

Entities are defined in OWL 2 QL in the same way as in the structural specification [*OWL 2 Specification*], and OWL 2 QL supports all predefined classes and properties. Furthermore, OWL 2 QL supports the following datatypes:

- *rdf:text*
- *rdf:XMLLiteral*
- *rdfs:Literal*
- *owl:real*
- *owl:rational*
- *xsd:decimal*
- *xsd:integer*
- *xsd:nonNegativeInteger*
- *xsd:string*
- *xsd:normalizedString*
- *xsd:token*
- *xsd:Name*
- *xsd:NCName*
- *xsd:NMTOKEN*
- *xsd:hexBinary*
- *xsd:base64Binary*
- *xsd:anyURI*
- *xsd:dateTime*
- *xsd:dateTimeStamp*

The set of supported datatypes has been designed such that the intersection of the value spaces of any set of these datatypes is either empty or infinite, which is necessary to obtain the desired computational properties. Consequently, the following datatypes *must not* be used in OWL 2 QL: *xsd:double*, *xsd:float*, *xsd:nonPositiveInteger*, *xsd:positiveInteger*, *xsd:negativeInteger*, *xsd:long*, *xsd:int*, *xsd:short*, *xsd:byte*, *xsd:unsignedLong*, *xsd:unsignedInt*, *xsd:unsignedShort*, *xsd:unsignedByte*, *xsd:language*, and *xsd:boolean*.

Finally, OWL 2 QL does not support anonymous individuals.

```
Individual := NamedIndividual
```

3.2.2 Property Expressions

OWL 2 QL object and data property expressions are the same as in the structural specification [*OWL 2 Specification*].

3.2.3 Class Expressions

In OWL 2 QL, there are two types of class expressions. The **subClassExpression** production defines the class expressions that can occur as subclass expressions in **SubClassOf** axioms, and the **superClassExpression** production defines the classes that can occur as superclass expressions in **SubClassOf** axioms.

```
subClassExpression :=
    Class |
    subObjectSomeValuesFrom | DataSomeValuesFrom
subObjectSomeValuesFrom := 'ObjectSomeValuesFrom' '('
ObjectPropertyExpression owl:Thing ')'

superClassExpression :=
    Class |
    superObjectIntersectionOf | superObjectComplementOf |
    superObjectSomeValuesFrom | DataSomeValuesFrom
superObjectIntersectionOf := 'ObjectIntersectionOf' '('
superClassExpression superClassExpression { superClassExpression }
')'
superObjectComplementOf := 'ObjectComplementOf' '('
subClassExpression ')'
superObjectSomeValuesFrom := 'ObjectSomeValuesFrom' '('
ObjectPropertyExpression Class ')'
```

W3C Editor's Draft

3.2.4 Data Ranges

A data range expression is restricted in OWL 2 QL to the predefined datatypes and the intersection of data ranges.

DataRange := **Datatype** | **DataIntersectionOf**

3.2.5 Axioms

The class axioms of OWL 2 QL are the same as in the structural specification [*OWL 2 Specification*], with the exception that **DisjointUnion** is disallowed; however, all axioms that refer to the **ClassExpression** production are redefined so as to use **subClassExpression** and/or **superClassExpression** as appropriate.

SubClassOf := 'SubClassOf' '(' **axiomAnnotations** **subClassExpression** **superClassExpression** ')'
EquivalentClasses := 'EquivalentClasses' '(' **axiomAnnotations** **subClassExpression** **subClassExpression** { **subClassExpression** } ')'
DisjointClasses := 'DisjointClasses' '(' **axiomAnnotations** **subClassExpression** **subClassExpression** { **subClassExpression** } ')'
ClassAxiom := **SubClassOf** | **EquivalentClasses** | **DisjointClasses**

OWL 2 QL disallows the use of property chains in property inclusion axioms; however, simple property inclusions are supported. Furthermore, OWL 2 QL disallows the use of functional and transitive object properties, and it restricts the class expressions in object property domain and range axioms to **superClassExpression**.

ObjectPropertyDomain := 'ObjectPropertyDomain' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'
ObjectPropertyRange := 'ObjectPropertyRange' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'
SubObjectPropertyOf := 'SubObjectPropertyOf' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** ')'
ObjectPropertyAxiom :=
SubObjectPropertyOf | **EquivalentObjectProperties** |
DisjointObjectProperties | **InverseObjectProperties** |
ObjectPropertyDomain | **ObjectPropertyRange** |

ReflexiveObjectProperty |
SymmetricObjectProperty | **AsymmetricObjectProperty**

OWL 2 QL disallows functional data property axioms, and it restricts the class expressions in data property domain axioms to **superClassExpression**.

DataPropertyDomain := 'DataPropertyDomain' '(' **axiomAnnotations** **DataPropertyExpression** **superClassExpression** ')'
DataPropertyAxiom :=
SubDataPropertyOf | **EquivalentDataProperties** | **DisjointDataProperties** |
DataPropertyDomain | **DataPropertyRange**

OWL 2 QL disallows negative object property assertions and individual equality axioms. Furthermore, class assertions in OWL 2 QL can involve only atomic classes. Inequality axioms and property assertions are the same as in the structural specification [*OWL 2 Specification*].

ClassAssertion := 'ClassAssertion' '(' **axiomAnnotations** **Class** **Individual** ')'
Assertion := **DifferentIndividuals** | **ClassAssertion** | **ObjectPropertyAssertion** | **DataPropertyAssertion**

Finally, the axioms in OWL 2 QL are the same as those in the structural specification [*OWL 2 Specification*], with the exception that key axioms are not allowed.

Axiom := **Declaration** | **ClassAxiom** | **ObjectPropertyAxiom** | **DataPropertyAxiom** | **DatatypeDefinition** | **Assertion** | **AnnotationAxiom**

4 OWL 2 RL

The OWL 2 RL profile is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade the full expressivity of the language for efficiency, and RDF(S) applications that need some added expressivity from OWL 2. This is achieved by defining a syntactic subset of OWL 2 which is amenable to implementation using rule-based technologies (see Section 4.2), and presenting a partial axiomatization of the OWL 2 RDF-Based Semantics [*OWL 2 RDF-Based Semantics*] in the form of first-order implications that can be used as the basis for such an implementation (see Section 4.3). The design of OWL 2 RL has been inspired by Description Logic Programs [*DLP*] and pD* [*pD**].

For ontologies satisfying the syntactic constraints described in Section 4.2, a suitable rule-based implementation (e.g., one based on the partial axiomatization presented in Section 4.3) will have desirable computational properties; for example, it can return *all* and *only* the correct answers to certain kinds of query (see Theorem PR1 and [*Conformance*]). Such an implementation can also be used with arbitrary RDF graphs. In this case, however, these properties no longer hold — in particular, it is no longer possible to guarantee that *all* correct answers can be returned, for example if the RDF graph uses the built-in vocabulary in unusual ways. Such an implementation will, however, still produce only correct entailments (see [*Conformance*]).

4.1 Feature Overview

Restricting the way in which constructs are used makes it possible to implement reasoning systems using rule-based reasoning engines, while still providing desirable computational guarantees. These restrictions are designed so as to avoid the need to infer the existence of individuals not explicitly present in the knowledge base, and to avoid the need for nondeterministic reasoning. This is achieved by restricting the use of constructs to certain syntactic positions. For example in `SubClassOf` axioms, the constructs in the subclass and superclass expressions must follow the usage patterns shown in Table 2.

Table 2. Syntactic Restrictions on Class Expressions in OWL 2 RL

Subclass Expressions	Superclass Expressions
a class other than *owl:Thing* an enumeration of individuals (**ObjectOneOf**) intersection of class expressions (**ObjectIntersectionOf**) union of class expressions (**ObjectUnionOf**) existential quantification to a class expression (**ObjectSomeValuesFrom**) existential quantification to a data range (**DataSomeValuesFrom**) existential quantification to an individual (**ObjectHasValue**) existential quantification to a literal (**DataHasValue**)	a class other than *owl:Thing* intersection of classes (**ObjectIntersectionOf**) negation (**ObjectComplementOf**) universal quantification to a class expression (**ObjectAllValuesFrom**) existential quantification to an individual (**ObjectHasValue**) at-most 0/1 cardinality restriction to a class expression (**ObjectMaxCardinality** 0/1) universal quantification to a data range (**DataAllValuesFrom**) existential quantification to a literal (**DataHasValue**) at-most 0/1 cardinality restriction to a data range (**DataMaxCardinality** 0/1)

All axioms in OWL 2 RL are constrained in a way that is compliant with these restrictions. Thus, OWL 2 RL supports all axioms of OWL 2 apart from disjoint unions of classes (**DisjointUnion**) and reflexive object property axioms (**ReflexiveObjectProperty**).

4.2 Profile Specification

The productions for OWL 2 RL are defined in the following sections. OWL 2 RL is defined not only in terms of the set of supported constructs, but it also restricts the places in which these constructs can be used. Note that each OWL 2 RL ontology must satisfy the global restrictions on axioms defined in Section 11 of the structural specification [*OWL 2 Specification*].

4.2.1 Entities

Entities are defined in OWL 2 RL in the same way as in the structural specification [*OWL 2 Specification*]. OWL 2 RL supports the predefined classes *owl:Nothing* and *owl:Thing*, but the usage of the latter class is restricted by the grammar of OWL 2 RL. Furthermore, OWL 2 RL does not support the predefined object and data properties *owl:topObjectProperty*, *owl:bottomObjectProperty*, *owl:topDataProperty*, and *owl:bottomDataProperty*. Finally, OWL 2 RL supports the following datatypes:

- *rdf:text*
- *rdf:XMLLiteral*
- *rdfs:Literal*
- *xsd:decimal*
- *xsd:integer*
- *xsd:nonNegativeInteger*
- *xsd:nonPositiveInteger*
- *xsd:positiveInteger*
- *xsd:negativeInteger*
- *xsd:long*
- *xsd:int*
- *xsd:short*
- *xsd:byte*
- *xsd:unsignedLong*
- *xsd:unsignedInt*
- *xsd:unsignedShort*
- *xsd:unsignedByte*
- *xsd:float*
- *xsd:double*
- *xsd:string*
- *xsd:normalizedString*
- *xsd:token*
- *xsd:language*
- *xsd:Name*
- *xsd:NCName*
- *xsd:NMTOKEN*

- *xsd:boolean*
- *xsd:hexBinary*
- *xsd:base64Binary*
- *xsd:anyURI*
- *xsd:dateTime*
- *xsd:dateTimeStamp*

The set of supported datatypes has been designed to allow for an implementation in rule systems. The *owl:real* and *owl:rational* datatypes *must not* be used in OWL 2 RL.

4.2.2 Property Expressions

Property expressions in OWL 2 RL are identical to the property expressions in the structural specification [*OWL 2 Specification*].

4.2.3 Class Expressions

There are three types of class expressions in OWL 2 RL. The **subClassExpression** production defines the class expressions that can occur as subclass expressions in **SubClassOf** axioms; the **superClassExpression** production defines the class expressions that can occur as superclass expressions in **SubClassOf** axioms; and the **equivClassExpressions** production defines the classes that can occur in **EquivalentClasses** axioms.

```
zeroOrOne := '0' | '1'

subClassExpression :=
    Class other than owl:Thing |
    subObjectIntersectionOf | subObjectUnionOf | ObjectOneOf |
    subObjectSomeValuesFrom | ObjectHasValue |
    DataSomeValuesFrom | DataHasValue
subObjectIntersectionOf := 'ObjectIntersectionOf' '('
subClassExpression subClassExpression { subClassExpression } ')'
subObjectUnionOf := 'ObjectUnionOf' '(' subClassExpression
subClassExpression { subClassExpression } ')'
subObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression
subClassExpression ')' |
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression
owl:Thing ')'

superClassExpression :=
    Class other than owl:Thing |
    superObjectIntersectionOf | superObjectComplementOf |
```

W3C Editor's Draft

```
    superObjectAllValuesFrom | ObjectHasValue |
superObjectMaxCardinality |
    DataAllValuesFrom | DataHasValue | superDataMaxCardinality
superObjectIntersectionOf := 'ObjectIntersectionOf' '('
superClassExpression superClassExpression { superClassExpression }
')'
superObjectComplementOf := 'ObjectComplementOf' '('
subClassExpression ')'
superObjectAllValuesFrom := 'ObjectAllValuesFrom' '('
ObjectPropertyExpression superClassExpression ')'
superObjectMaxCardinality :=
    'ObjectMaxCardinality' '(' zeroOrOne
ObjectPropertyExpression [ subClassExpression ] ')' |
    'ObjectMaxCardinality' '(' zeroOrOne
ObjectPropertyExpression owl:Thing ')'
superDataMaxCardinality := 'DataMaxCardinality' '(' zeroOrOne
DataPropertyExpression [ DataRange ] ')'

equivClassExpression :=
    Class other than owl:Thing |
    equivObjectIntersectionOf |
    ObjectHasValue |
    DataHasValue
equivObjectIntersectionOf := 'ObjectIntersectionOf' '('
equivClassExpression equivClassExpression { equivClassExpression }
')'
```

4.2.4 Data Ranges

A data range expression is restricted in OWL 2 RL to the predefined datatypes admitted in OWL 2 RL and the intersection of data ranges.

```
DataRange := Datatype | DataIntersectionOf
```

4.2.5 Axioms

OWL 2 RL redefines all axioms of the structural specification [*OWL 2 Specification*] that refer to class expressions. In particular, it restricts various class axioms to use the appropriate form of class expressions (i.e., one of **subClassExpression**, **superClassExpression**, or **equivClassExpression**), and it disallows the **DisjointUnion** axiom.

```
ClassAxiom := SubClassOf | EquivalentClasses | DisjointClasses
SubClassOf := 'SubClassOf' '(' axiomAnnotations
subClassExpression superClassExpression ')'
EquivalentClasses := 'EquivalentClasses' '(' axiomAnnotations
equivClassExpression equivClassExpression { equivClassExpression }
')'
DisjointClasses := 'DisjointClasses' '(' axiomAnnotations
subClassExpression subClassExpression { subClassExpression } ')'
```

OWL 2 RL axioms about property expressions are as in the structural specification [OWL 2 Specification], the only differences being that class expressions in property domain and range axioms are restricted to **superClassExpression**, and that the use of reflexive properties is disallowed.

```
ObjectPropertyDomain := 'ObjectPropertyDomain' '('
axiomAnnotations ObjectPropertyExpression superClassExpression ')'
ObjectPropertyRange := 'ObjectPropertyRange' '('
axiomAnnotations ObjectPropertyExpression superClassExpression ')'
DataPropertyDomain := 'DataPropertyDomain' '(' axiomAnnotations
DataPropertyExpression superClassExpression ')'
ObjectPropertyAxiom :=
    SubObjectPropertyOf | EquivalentObjectProperties |
    DisjointObjectProperties | InverseObjectProperties |
    ObjectPropertyDomain | ObjectPropertyRange |
    FunctionalObjectProperty | InverseFunctionalObjectProperty |
    IrreflexiveObjectProperty |
    SymmetricObjectProperty | AsymmetricObjectProperty
    TransitiveObjectProperty
```

OWL 2 RL restricts class expressions in positive assertions to **superClassExpression**. All other assertions are the same as in the structural specification [OWL 2 Specification].

```
ClassAssertion := 'ClassAssertion' '(' axiomAnnotations
superClassExpression Individual ')'
```

OWL 2 RL restricts class expressions in keys to **subClassExpression**.

HasKey := 'HasKey' '(' **axiomAnnotations** **subClassExpression** '(' { **ObjectPropertyExpression** } ')' '(' { **DataPropertyExpression** } ')' ')'

All other axioms in OWL 2 RL are defined as in the structural specification [*OWL 2 Specification*].

4.3 Reasoning in OWL 2 RL and RDF Graphs using Rules

This section presents a partial axiomatization of the OWL 2 RDF-Based Semantics [*OWL 2 RDF-Based Semantics*] in the form of first-order implications; this axiomatization is called the OWL 2 RL/RDF rules. These rules provide a useful starting point for practical implementation using rule-based technologies such as logic programming [*Logic Programming*].

The rules are given as universally quantified first-order implications over a ternary predicate `T`. This predicate represents a generalization of RDF triples in which bnodes and literals are allowed in all positions (similar to the partial generalization in pD* [*pD**] and to generalized RDF triples in RIF [*RIF RDF & OWL*]); thus, `T(s, p, o)` represents a generalized RDF triple with the subject `s`, predicate `p`, and the object `o`. Variables in the implications are preceded with a question mark. The rules that have empty "if" parts should be understood as being always applicable. The propositional symbol `false` is a special symbol denoting contradiction: if it is derived, then the initial RDF graph was inconsistent. The set of rules listed in this section is not minimal, as certain rules are implied by other ones; this was done to make the definition of the semantic consequences of each piece of OWL 2 vocabulary self-contained.

Many conditions contain atoms that match to the list construct of RDF. In order to simplify the presentation of the rules, LIST[h, e_1, ..., e_n] is used as an abbreviation for the conjunction of triples shown in Table 3, where z_2, ..., z_n are fresh variables that do not occur anywhere where the abbreviation is used.

Table 3. Expansion of LIST[h, e_1, ..., e_n]

T(h, rdf:first, e_1)	T(h, rdf:rest, z_2)
T(z_2, rdf:first, e_2)	T(z_2, rdf:rest, z_3)
...	...
T(z_n, rdf:first, e_n)	T(z_n, rdf:rest, rdf:nil)

W3C Editor's Draft

The axiomatization is split into several tables for easier navigation. Each rule is given a short unique name. The rows of several tables specify rules that need to be instantiated for each combination of indices given in the right-most column.

Table 4 axiomatizes the semantics of equality. In particular, it defines the equality relation `owl:sameAs` as being reflexive, symmetric, and transitive, and it axiomatizes the standard replacement properties of equality for it.

Table 4. The Semantics of Equality

	If	then	
eq-ref	T(?s, ?p, ?o)	T(?s, owl:sameAs, ?s) T(?p, owl:sameAs, ?p) T(?o, owl:sameAs, ?o)	
eq-sym	T(?x, owl:sameAs, ?y)	T(?y, owl:sameAs, ?x)	
eq-trans	T(?x, owl:sameAs, ?y) T(?y, owl:sameAs, ?z)	T(?x, owl:sameAs, ?z)	
eq-rep-s	T(?s, owl:sameAs, ?s') T(?s, ?p, ?o)	T(?s', ?p, ?o)	
eq-rep-p	T(?p, owl:sameAs, ?p') T(?s, ?p, ?o)	T(?s, ?p', ?o)	
eq-rep-o	T(?o, owl:sameAs, ?o') T(?s, ?p, ?o)	T(?s, ?p, ?o')	
eq-diff1	T(?x, owl:sameAs, ?y) T(?x, owl:differentFrom, ?y)	false	
eq-diff2	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:members, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each 1 ≤ i < j ≤ n
eq-diff3	T(?x, rdf:type, owl:AllDifferent) T(?x, owl:distinctMembers, ?y) LIST[?y, $?z_1$, ..., $?z_n$] T($?z_i$, owl:sameAs, $?z_j$)	false	for each 1 ≤ i < j ≤ n

Table 5 specifies the semantic conditions on axioms about properties.

Table 5. The Semantics of Axioms about Properties

	If	then	
prp-ap		T(ap, rdf:type, owl:AnnotationProperty)	for each built-in annotation property of OWL 2 RL
prp-dom	T(?p, rdfs:domain, ?c) T(?x, ?p, ?y)	T(?x, rdf:type, ?c)	
prp-rng	T(?p, rdfs:range, ?c) T(?x, ?p, ?y)	T(?y, rdf:type, ?c)	
prp-fp	T(?p, rdf:type, owl:FunctionalProperty) T(?x, ?p, $?y_1$) T(?x, ?p, $?y_2$)	T($?y_1$, owl:sameAs, $?y_2$)	
prp-ifp	T(?p, rdf:type, owl:InverseFunctionalProperty) T($?x_1$, ?p, ?y) T($?x_2$, ?p, ?y)	T($?x_1$, owl:sameAs, $?x_2$)	
prp-irp	T(?p, rdf:type, owl:IrreflexiveProperty) T(?x, ?p, ?x)	false	
prp-symp	T(?p, rdf:type, owl:SymmetricProperty) T(?x, ?p, ?y)	T(?y, ?p, ?x)	
prp-asyp	T(?p, rdf:type, owl:AsymmetricProperty) T(?x, ?p, ?y) T(?y, ?p, ?x)	false	
prp-trp	T(?p, rdf:type, owl:TransitiveProperty) T(?x, ?p, ?y) T(?y, ?p, ?z)	T(?x, ?p, ?z)	
prp-spo1	T($?p_1$, rdfs:subPropertyOf, $?p_2$) T(?x, $?p_1$, ?y)	T(?x, $?p_2$, ?y)	

prp-spo2	T(?p, owl:propertyChainAxiom, ?x) LIST[?x, ?p$_1$, ..., ?p$_n$] T(?u$_1$, ?p$_1$, ?u$_2$) T(?u$_2$, ?p$_2$, ?u$_3$) ... T(?u$_n$, ?p$_n$, ?u$_{n+1}$)	T(?u$_1$, ?p, ?u$_{n+1}$)	
prp-eqp1	T(?p$_1$, owl:equivalentProperty, ?p$_2$) T(?x, ?p$_1$, ?y)	T(?x, ?p$_2$, ?y)	
prp-eqp2	T(?p$_1$, owl:equivalentProperty, ?p$_2$) T(?x, ?p$_2$, ?y)	T(?x, ?p$_1$, ?y)	
prp-pdw	T(?p$_1$, owl:propertyDisjointWith, ?p$_2$) T(?x, ?p$_1$, ?y) T(?x, ?p$_2$, ?y)	false	
prp-adp	T(?x, rdf:type, owl:AllDisjointProperties) T(?x, owl:members, ?y) LIST[?y, ?p$_1$, ..., ?p$_n$] T(?u, ?p$_i$, ?v) T(?u, ?p$_j$, ?v)	false	for each $1 \le i < j \le n$
prp-inv1	T(?p$_1$, owl:inverseOf, ?p$_2$) T(?x, ?p$_1$, ?y)	T(?y, ?p$_2$, ?x)	
prp-inv2	T(?p$_1$, owl:inverseOf, ?p$_2$) T(?x, ?p$_2$, ?y)	T(?y, ?p$_1$, ?x)	
prp-key	T(?c, owl:hasKey, ?u) LIST[?u, ?p$_1$, ..., ?p$_n$] T(?x, rdf:type, ?c) T(?x, ?p$_1$, ?z$_1$) ... T(?x, ?p$_n$, ?z$_n$) T(?y, rdf:type, ?c) T(?y, ?p$_1$, ?z$_1$) ... T(?y, ?p$_n$, ?z$_n$)	T(?x, owl:sameAs, ?y)	
prp-npa1	T(?x, owl:sourceIndividual, ?i$_1$) T(?x, owl:assertionProperty, ?p) T(?x,	false	

	owl:targetIndividual, ?i$_2$) T(?i$_1$, ?p, ?i$_2$)		
prp-npa2	T(?x, owl:sourceIndividual, ?i) T(?x, owl:assertionProperty, ?p) T(?x, owl:targetValue, ?lt) T(?i, ?p, ?lt)	false	

Table 6 specifies the semantic conditions on classes.

Table 6. The Semantics of Classes

	If	then	
cls-thing		T(owl:Thing, rdf:type, owl:Class)	
cls-nothing1		T(owl:Nothing, rdf:type, owl:Class)	
cls-nothing2	T(?x, rdf:type, owl:Nothing)	false	
cls-int1	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c$_1$, ..., ?c$_n$] T(?y, rdf:type, ?c$_1$) T(?y, rdf:type, ?c$_2$) ... T(?y, rdf:type, ?c$_n$)	T(?y, rdf:type, ?c)	
cls-int2	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c$_1$, ..., ?c$_n$] T(?y, rdf:type, ?c)	T(?y, rdf:type, ?c$_1$) T(?y, rdf:type, ?c$_2$) ... T(?y, rdf:type, ?c$_n$)	
cls-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c$_1$, ..., ?c$_n$] T(?y, rdf:type, ?c$_i$)	T(?y, rdf:type, ?c)	for each $1 \leq i \leq n$
cls-com	T(?c$_1$, owl:complementOf, ?c$_2$)	false	

W3C Editor's Draft

	T(?x, rdf:type, ?c1) T(?x, rdf:type, ?c2)	
cls-svf1	T(?x, owl:someValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v) T(?v, rdf:type, ?y)	T(?u, rdf:type, ?x)
cls-svf2	T(?x, owl:someValuesFrom, owl:Thing) T(?x, owl:onProperty, ?p) T(?u, ?p, ?v)	T(?u, rdf:type, ?x)
cls-avf	T(?x, owl:allValuesFrom, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?v)	T(?v, rdf:type, ?y)
cls-hv1	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x)	T(?u, ?p, ?y)
cls-hv2	T(?x, owl:hasValue, ?y) T(?x, owl:onProperty, ?p) T(?u, ?p, ?y)	T(?u, rdf:type, ?x)
cls-maxc1	T(?x, owl:maxCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxc2	T(?x, owl:maxCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?u, rdf:type, ?x) T(?u, ?p, ?y1) T(?u, ?p, ?y2)	T(?y1, owl:sameAs, ?y2)
cls-maxqc1	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, ?y) T(?y, rdf:type, ?c)	false

cls-maxqc2	T(?x, owl:maxQualifiedCardinality, "0"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, ?y)	false
cls-maxqc3	T(?x, owl:maxQualifiedCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, ?c) T(?u, rdf:type, ?x) T(?u, ?p, ?y_1) T(?y_1, rdf:type, ?c) T(?u, ?p, ?y_2) T(?y_2, rdf:type, ?c)	T(?y_1, owl:sameAs, ?y_2)
cls-maxqc4	T(?x, owl:maxQualifiedCardinality, "1"^^xsd:nonNegativeInteger) T(?x, owl:onProperty, ?p) T(?x, owl:onClass, owl:Thing) T(?u, rdf:type, ?x) T(?u, ?p, ?y_1) T(?u, ?p, ?y_2)	T(?y_1, owl:sameAs, ?y_2)
cls-oo	T(?c, owl:oneOf, ?x) LIST[?x, ?y_1, ..., ?y_n]	T(?y_1, rdf:type, ?c) ... T(?y_n, rdf:type, ?c)

Table 7 specifies the semantic conditions on class axioms.

Table 7. The Semantics of Class Axioms

	If	then
cax-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)
cax-eqc1	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)

cax-eqc2	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_2)	T(?x, rdf:type, ?c_1)	
cax-dw	T(?c_1, owl:disjointWith, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false	
cax-adc	T(?x, rdf:type, owl:AllDisjointClasses) T(?x, owl:members, ?y) LIST[?y, ?c_1, ..., ?c_n] T(?z, rdf:type, ?c_i) T(?z, rdf:type, ?c_j)	false	for each $1 \le i < j \le n$

Table 8 specifies the semantics of datatypes.

Table 8. The Semantics of Datatypes

	If	then	
dt-type1		T(dt, rdf:type, rdfs:Datatype)	for each datatype dt supported in OWL 2 RL
dt-type2		T(lt, rdf:type, dt)	for each literal lt and each datatype dt supported in OWL 2 RL such that the data value of lt is contained in the value space of dt
dt-eq		T(lt_1, owl:sameAs, lt_2)	for all literals lt_1 and lt_2 with the same data value
dt-diff		T(lt_1, owl:differentFrom, lt_2)	for all literals lt_1 and lt_2 with different data values
dt-not-type	T(lt, rdf:type, dt)	false	for each literal lt and each datatype dt supported in OWL 2 RL such that the data value of lt is not contained in the value space of dt

Table 9 specifies the semantic restrictions on the vocabulary used to define the schema.

Table 9. The Semantics of Schema Vocabulary

	If	then
scm-cls	T(?c, rdf:type, owl:Class)	T(?c, rdfs:subClassOf, ?c) T(?c, owl:equivalentClass, ?c) T(?c, rdfs:subClassOf, owl:Thing) T(owl:Nothing, rdfs:subClassOf, ?c)
scm-sco	T(?c$_1$, rdfs:subClassOf, ?c$_2$) T(?c$_2$, rdfs:subClassOf, ?c$_3$)	T(?c$_1$, rdfs:subClassOf, ?c$_3$)
scm-eqc1	T(?c$_1$, owl:equivalentClass, ?c$_2$)	T(?c$_1$, rdfs:subClassOf, ?c$_2$) T(?c$_2$, rdfs:subClassOf, ?c$_1$)
scm-eqc2	T(?c$_1$, rdfs:subClassOf, ?c$_2$) T(?c$_2$, rdfs:subClassOf, ?c$_1$)	T(?c$_1$, owl:equivalentClass, ?c$_2$)
scm-op	T(?p, rdf:type, owl:ObjectProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-dp	T(?p, rdf:type, owl:DatatypeProperty)	T(?p, rdfs:subPropertyOf, ?p) T(?p, owl:equivalentProperty, ?p)
scm-spo	T(?p$_1$, rdfs:subPropertyOf, ?p$_2$) T(?p$_2$, rdfs:subPropertyOf, ?p$_3$)	T(?p$_1$, rdfs:subPropertyOf, ?p$_3$)
scm-eqp1	T(?p$_1$, owl:equivalentProperty, ?p$_2$)	T(?p$_1$, rdfs:subPropertyOf, ?p$_2$) T(?p$_2$, rdfs:subPropertyOf, ?p$_1$)
scm-eqp2	T(?p$_1$, rdfs:subPropertyOf, ?p$_2$) T(?p$_2$, rdfs:subPropertyOf, ?p$_1$)	T(?p$_1$, owl:equivalentProperty, ?p$_2$)
scm-dom1	T(?p, rdfs:domain, ?c$_1$) T(?c$_1$, rdfs:subClassOf, ?c$_2$)	T(?p, rdfs:domain, ?c$_2$)

scm-dom2	T(?p_2, rdfs:domain, ?c) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?p_1, rdfs:domain, ?c)
scm-rng1	T(?p, rdfs:range, ?c_1) T(?c_1, rdfs:subClassOf, ?c_2)	T(?p, rdfs:range, ?c_2)
scm-rng2	T(?p_2, rdfs:range, ?c) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?p_1, rdfs:range, ?c)
scm-hv	T(?c_1, owl:hasValue, ?i) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:hasValue, ?i) T(?c_2, owl:onProperty, ?p_2) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-svf1	T(?c_1, owl:someValuesFrom, ?y_1) T(?c_1, owl:onProperty, ?p) T(?c_2, owl:someValuesFrom, ?y_2) T(?c_2, owl:onProperty, ?p) T(?y_1, rdfs:subClassOf, ?y_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-svf2	T(?c_1, owl:someValuesFrom, ?y) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:someValuesFrom, ?y) T(?c_2, owl:onProperty, ?p_2) T(?p_1, rdfs:subPropertyOf, ?p_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-avf1	T(?c_1, owl:allValuesFrom, ?y_1) T(?c_1, owl:onProperty, ?p) T(?c_2, owl:allValuesFrom, ?y_2) T(?c_2, owl:onProperty, ?p) T(?y_1, rdfs:subClassOf, ?y_2)	T(?c_1, rdfs:subClassOf, ?c_2)
scm-avf2	T(?c_1, owl:allValuesFrom, ?y) T(?c_1, owl:onProperty, ?p_1) T(?c_2, owl:allValuesFrom, ?y) T(?c_2, owl:onProperty, ?p_2)	T(?c_2, rdfs:subClassOf, ?c_1)

W3C Editor's Draft

	T(?p1, rdfs:subPropertyOf, ?p2)	
scm-int	T(?c, owl:intersectionOf, ?x) LIST[?x, ?c1, ..., ?cn]	T(?c, rdfs:subClassOf, ?c1) T(?c, rdfs:subClassOf, ?c2) ... T(?c, rdfs:subClassOf, ?cn)
scm-uni	T(?c, owl:unionOf, ?x) LIST[?x, ?c1, ..., ?cn]	T(?c1, rdfs:subClassOf, ?c) T(?c2, rdfs:subClassOf, ?c) ... T(?cn, rdfs:subClassOf, ?c)

In order to avoid potential performance problems in practice, OWL 2 RL/RDF rules do not include the axiomatic triples of RDF and RDFS (i.e., those triples that must be satisfied by, respectively, every RDF and RDFS interpretation) [RDF Semantics] and the relevant OWL vocabulary [OWL 2 RDF-Based Semantics]; moreover, OWL 2 RL/RDF rules include most, but not all of the entailment rules of RDFS [RDF Semantics]. An OWL 2 RL/RDF implementation *may* include these triples and entailment rules as necessary without invalidating the conformance requirements for OWL 2 RL [Conformance].

Theorem PR1. Let *R* be the OWL 2 RL/RDF rules as defined above. Furthermore, let O_1 and O_2 be OWL 2 RL ontologies satisfying the following properties:

- neither O_1 nor O_2 contains a IRI that is used for more than one type of entity (i.e., no IRIs is used both as, say, a class and an individual);
- O_1 does not contain **SubAnnotationPropertyOf**, **AnnotationPropertyDomain**, and **AnnotationPropertyRange** axioms; and
- each axiom in O_2 is an assertion of the form as specified below, for `a`, a_1, ..., a_n named individuals:
  - `ClassAssertion( C a )` where `C` is a class,
  - `ObjectPropertyAssertion( OP a1 a2 )` where `OP` is an object property,
  - `DataPropertyAssertion( DP a v )` where `DP` is a data property, or
  - `SameIndividual( a1 ... an )`.

Furthermore, let $RDF(O_1)$ and $RDF(O_2)$ be translations of O_1 and O_2, respectively, into RDF graphs as specified in the OWL 2 Mapping to RDF Graphs [OWL 2 RDF Mapping]; and let $FO(RDF(O_1))$ and $FO(RDF(O_2))$ be the translation of these graphs into first-order theories in which triples are represented using the `T` predicate — that is, `T(s, p, o)` represents an RDF triple with the subject `s`, predicate `p`, and the object `o`. Then, O_1 entails O_2 under the OWL 2 Direct Semantics [OWL 2 Direct Semantics] if and only if $FO(RDF(O_1)) \cup R$ entails $FO(RDF(O_2))$ under the standard first-order semantics.

W3C Editor's Draft

Proof Sketch. Without loss of generality, it can be assumed that all axioms in O_1 are fully normalized — that is, that all class expressions in the axioms are of depth at most one. Let $DLP(O_1)$ be the set of rules obtained by translating O_1 into a set of rules as in Description Logic Programs [DLP].

Consider now each assertion $A \in O_2$ that is entailed by $DLP(O_1)$ (or, equivalently, by O_1). Let *dt* be a derivation tree for *A* from $DLP(O_1)$. By examining the set of OWL 2 RL constructs, it is possible to see that each such tree can be transformed to a derivation tree *dt'* for *FO(RDF(A))* from $FO(RDF(O_1)) \cup R$. Each assertion *B* occurring in *dt* is of the form as specified in the theorem. The tree *dt'* can, roughly speaking, be obtained from *dt* by replacing each assertion *B* with *FO(RDF(B))* and by replacing each rule from $DLP(O_1)$ with a corresponding rule from Tables 3–8. Consequently, $FO(RDF(O_1)) \cup R$ entails *FO(RDF(A))*.

Since no IRI in O_1 is used as both an individual and a class or a property, $FO(RDF(O_1)) \cup R$ does not entail a triple of the form `T(a:i1, owl:sameAs, a:i2)` where either *a:i1* or *a:i2* is used in O_1 as a class or a property. This allows one to transform a derivation tree for *FO(RDF(A))* from $FO(RDF(O_1)) \cup R$ to a derivation tree for *A* from $DLP(O_1)$ in a way that is analogous to the previous case. QED

5 Computational Properties

This section describes the computational complexity of the most relevant reasoning problems of the languages defined in this document. For an introduction to computational complexity, please refer to a textbook on complexity such as [Papadimitriou]. The reasoning problems considered here are *ontology consistency*, *class expression satisfiability*, *class expression subsumption*, *instance checking*, and *(Boolean) conjunctive query answering* [OWL 2 Direct Semantics]. When evaluating complexity, the following parameters will be considered:

- **Data Complexity**: the complexity measured with respect to the total size of the assertions in the ontology.
- **Taxonomic Complexity**: the complexity measured with respect to the total size of the axioms in the ontology.
- **Query Complexity**: the complexity measured with respect to the total size of the query.
- **Combined Complexity**: the complexity measured with respect to both the size of the axioms, the size of the assertions, and, in the case of conjunctive query answering, the size of the query as well.

Table 10 summarizes the known complexity results for OWL 2 under both RDF and the direct semantics, OWL 2 EL, OWL 2 QL, OWL 2 RL, and OWL 1 DL. The meaning of the entries is as follows:

- **Decidability open** means that it is not known whether this reasoning problem is decidable at all.

- **Decidable, but complexity open** means that decidability of this reasoning problem is known, but not its exact computational complexity. If available, known lower bounds are given in parenthesis; for example, **(NP-Hard)** means that this problem is at least as hard as any other problem in NP.
- **X-complete** for X one of the complexity classes explained below indicates that tight complexity bounds are known — that is, the problem is known to be both *in* the complexity class X (i.e., an algorithm is known that only uses time/space in X) and *hard* for X (i.e., it is at least as hard as any other problem in X). The following is a brief sketch of the classes used in this table, from the most complex one down to the simplest ones.
    - **2NEXPTIME** is the class of problems solvable by a nondeterministic algorithm in *time* that is at most double exponential in the size of the input (i.e., roughly 2^{2^n}, for n the size of the input).
    - **NEXPTIME** is the class of problems solvable by a nondeterministic algorithm in *time* that is at most exponential in the size of the input (i.e., roughly 2^n, for n the size of the input).
    - **PSPACE** is the class of problems solvable by a deterministic algorithm using *space* that is at most polynomial in the size of the input (i.e., roughly n^c, for n the size of the input and c a constant).
    - **NP** is the class of problems solvable by a nondeterministic algorithm using *time* that is at most polynomial in the size of the input (i.e., roughly n^c, for n the size of the input and c a constant).
    - **PTIME** is the class of problems solvable by a deterministic algorithm using *time* that is at most polynomial in the size of the input (i.e., roughly n^c, for n the size of the input and c a constant). PTIME is often referred to as *tractable*, whereas the problems in the classes above are often referred to as *intractable*.
    - **LOGSPACE** is the class of problems solvable by a deterministic algorithm using *space* that is at most logarithmic in the size of the input (i.e., roughly log(n), for n the size of the input and c a constant). NLOGSPACE is the nondeterministic version of this class.
    - **AC^0** is a proper subclass of LOGSPACE and defined not via Turing Machines, but via circuits: AC^0 is the class of problems definable using a family of circuits of constant depth and polynomial size, which can be generated by a deterministic Turing machine in logarithmic time (in the size of the input). Intuitively, AC^0 allows us to use polynomially many processors but the run-time must be constant. A typical example of an AC^0 problem is the evaluation of first-order queries over databases (or model checking of first-order sentences over finite models), where only the database (first-order model) is regarded as the input and the query (first-order sentence) is assumed to be fixed. The undirected graph reachability problem is known to be in LogSpace, but not in AC^0.

The results below refer to the *worst-case* complexity of these reasoning problems and, as such, do not say that implemented algorithms necessarily run in this class on all input problems, or what space/time they use on some/typical/certain kind of problems. For X-complete problems, these results only say that a reasoning algorithm cannot use less time/space than indicated by this class on *all* input problems.

Table 10. Complexity of the Profiles

Language	Reasoning Problems	Taxonomic Complexity	Data Complexity	Query Complexity	Combined Complexity
OWL 2 RDF-Based Semantics	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking, Conjunctive Query Answering	Undecidable	Undecidable	Undecidable	Undecidable
OWL 2 Direct Semantics	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking	2NEXPTIME-complete (NEXPTIME if property hierarchies are bounded)	Decidable, but complexity open (NP-Hard)	Not Applicable	2NEXPTIME-complete (NEXPTIME if property hierarchies are bounded)
	Conjunctive Query Answering	Decidability open	Decidability open	Decidability open	Decidability open

OWL 2 EL	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking	PTIME-complete	PTIME-complete	Not Applicable	PTIME-complete
	Conjunctive Query Answering	PTIME-complete	PTIME-complete	NP-complete	PSPACE-complete
OWL 2 QL	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking,	NLogSpace-complete	In AC^0	Not Applicable	NLogSpace-complete
	Conjunctive Query Answering	NLogSpace-complete	In AC^0	NP-complete	NP-complete
OWL 2 RL	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking	PTIME-complete	PTIME-complete	Not Applicable	PTIME-complete
	Conjunctive Query Answering	PTIME-complete	PTIME-complete	NP-complete	NP-complete

OWL 1 DL	Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, Instance Checking	NEXPTIME-complete	Decidable, but complexity open (NP-Hard)	Not Applicable	NEXPTIME-complete
	Conjunctive Query Answering	Decidability open	Decidability open	Decidability open	Decidability open

6 Appendix: Complete Grammars for Profiles

This appendix contains the grammars for all three profiles of OWL 2.

6.1 OWL 2 EL

The grammar of OWL 2 EL consists of the general definitions from Section 13.1 of the OWL 2 Specification [*OWL 2 Specification*], as well as the following productions.

```
Class := IRI

Datatype := IRI

ObjectProperty := IRI

DataProperty := IRI

AnnotationProperty := IRI

Individual := NamedIndividual

NamedIndividual := IRI

Literal := typedLiteral | stringLiteralNoLanguage |
stringLiteralWithLanguage
typedLiteral := lexicalForm '^^' Datatype
lexicalForm := quotedString
stringLiteralNoLanguage := quotedString
```

```
stringLiteralWithLanguage := quotedString languageTag


ObjectPropertyExpression := ObjectProperty

DataPropertyExpression := DataProperty


DataRange := Datatype | DataIntersectionOf | DataOneOf

DataIntersectionOf := 'DataIntersectionOf' '(' DataRange
DataRange { DataRange } ')'

DataOneOf := 'DataOneOf' '(' Literal ')'


ClassExpression :=
    Class | ObjectIntersectionOf | ObjectOneOf |
    ObjectSomeValuesFrom | ObjectHasValue | ObjectHasSelf |
    DataSomeValuesFrom | DataHasValue

ObjectIntersectionOf := 'ObjectIntersectionOf' '(' ClassExpression
ClassExpression { ClassExpression } ')'

ObjectOneOf := 'ObjectOneOf' '(' Individual ')'

ObjectSomeValuesFrom := 'ObjectSomeValuesFrom' '('
ObjectPropertyExpression ClassExpression ')'

ObjectHasValue := 'ObjectHasValue' '(' ObjectPropertyExpression
Individual ')'

ObjectHasSelf := 'ObjectHasSelf' '(' ObjectPropertyExpression ')'

DataSomeValuesFrom := 'DataSomeValuesFrom' '('
DataPropertyExpression { DataPropertyExpression } DataRange ')'

DataHasValue := 'DataHasValue' '(' DataPropertyExpression Literal
')'


Axiom := Declaration | ClassAxiom | ObjectPropertyAxiom |
```

W3C Editor's Draft

```
DataPropertyAxiom | DatatypeDefinition | HasKey | Assertion |
AnnotationAxiom



ClassAxiom := SubClassOf | EquivalentClasses | DisjointClasses
SubClassOf := 'SubClassOf' '(' axiomAnnotations
subClassExpression superClassExpression ')'
subClassExpression := ClassExpression
superClassExpression := ClassExpression

EquivalentClasses := 'EquivalentClasses' '(' axiomAnnotations
ClassExpression ClassExpression { ClassExpression } ')'

DisjointClasses := 'DisjointClasses' '(' axiomAnnotations
ClassExpression ClassExpression { ClassExpression } ')'



ObjectPropertyAxiom :=
    EquivalentObjectProperties | SubObjectPropertyOf |
    ObjectPropertyDomain | ObjectPropertyRange |
    ReflexiveObjectProperty | TransitiveObjectProperty

SubObjectPropertyOf := 'SubObjectPropertyOf' '('
axiomAnnotations subObjectPropertyExpression
superObjectPropertyExpression ')'
subObjectPropertyExpression := ObjectPropertyExpression |
propertyExpressionChain
propertyExpressionChain := 'ObjectPropertyChain' '('
ObjectPropertyExpression ObjectPropertyExpression {
ObjectPropertyExpression } ')'
superObjectPropertyExpression := ObjectPropertyExpression

EquivalentObjectProperties := 'EquivalentObjectProperties' '('
axiomAnnotations ObjectPropertyExpression ObjectPropertyExpression {
ObjectPropertyExpression } ')'

ObjectPropertyDomain := 'ObjectPropertyDomain' '('
axiomAnnotations ObjectPropertyExpression ClassExpression ')'

ObjectPropertyRange := 'ObjectPropertyRange' '('
axiomAnnotations ObjectPropertyExpression ClassExpression ')'

ReflexiveObjectProperty := 'ReflexiveObjectProperty' '('
axiomAnnotations ObjectPropertyExpression ')'
```

TransitiveObjectProperty := 'TransitiveObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

DataPropertyAxiom :=
 SubDataPropertyOf | **EquivalentDataProperties** |
 DataPropertyDomain | **DataPropertyRange** | **FunctionalDataProperty**

SubDataPropertyOf := 'SubDataPropertyOf' '(' **axiomAnnotations** **subDataPropertyExpression** **superDataPropertyExpression** ')'
subDataPropertyExpression := **DataPropertyExpression**
superDataPropertyExpression := **DataPropertyExpression**

EquivalentDataProperties := 'EquivalentDataProperties' '(' **axiomAnnotations** **DataPropertyExpression** **DataPropertyExpression** { **DataPropertyExpression** } ')'

DataPropertyDomain := 'DataPropertyDomain' '(' **axiomAnnotations** **DataPropertyExpression** **ClassExpression** ')'

DataPropertyRange := 'DataPropertyRange' '(' **axiomAnnotations** **DataPropertyExpression** **DataRange** ')'

FunctionalDataProperty := 'FunctionalDataProperty' '(' **axiomAnnotations** **DataPropertyExpression** ')'

DatatypeDefinition := 'DatatypeDefinition' '(' **axiomAnnotations** **Datatype** **DataRange** ')'

HasKey := 'HasKey' '(' **axiomAnnotations** **ClassExpression** '(' { **ObjectPropertyExpression** } ')' '(' { **DataPropertyExpression** } ')' ')'

Assertion :=
 SameIndividual | **DifferentIndividuals** | **ClassAssertion** |
 ObjectPropertyAssertion | **NegativeObjectPropertyAssertion** |
 DataPropertyAssertion | **NegativeDataPropertyAssertion**

W3C Editor's Draft

```
sourceIndividual := Individual
targetIndividual := Individual
targetValue := Literal

SameIndividual := 'SameIndividual' '(' axiomAnnotations Individual
Individual { Individual } ')'

DifferentIndividuals := 'DifferentIndividuals' '(' axiomAnnotations
Individual Individual { Individual } ')'

ClassAssertion := 'ClassAssertion' '(' axiomAnnotations
ClassExpression Individual ')'

ObjectPropertyAssertion := 'ObjectPropertyAssertion' '('
axiomAnnotations ObjectPropertyExpression sourceIndividual
targetIndividual ')'

NegativeObjectPropertyAssertion :=
'NegativeObjectPropertyAssertion' '(' axiomAnnotations
ObjectPropertyExpression sourceIndividual targetIndividual ')'

DataPropertyAssertion := 'DataPropertyAssertion' '('
axiomAnnotations DataPropertyExpression sourceIndividual targetValue
')'

NegativeDataPropertyAssertion := 'NegativeDataPropertyAssertion'
'(' axiomAnnotations DataPropertyExpression sourceIndividual
targetValue ')'
```

6.2 OWL 2 QL

The grammar of OWL 2 QL consists of the general definitions from Section 13.1 of the OWL 2 Specification [*OWL 2 Specification*], as well as the following productions.

```
Class := IRI

Datatype := IRI

ObjectProperty := IRI

DataProperty := IRI
```

W3C Editor's Draft

AnnotationProperty := **IRI**

Individual := **NamedIndividual**

NamedIndividual := **IRI**

Literal := **typedLiteral** | **stringLiteralNoLanguage** | **stringLiteralWithLanguage**
typedLiteral := **lexicalForm** '^^' **Datatype**
lexicalForm := **quotedString**
stringLiteralNoLanguage := **quotedString**
stringLiteralWithLanguage := **quotedString** **languageTag**

ObjectPropertyExpression := **ObjectProperty** | **InverseObjectProperty**

InverseObjectProperty := 'ObjectInverseOf' '(' **ObjectProperty** ')'

DataPropertyExpression := **DataProperty**

DataRange := **Datatype** | **DataIntersectionOf**

DataIntersectionOf := 'DataIntersectionOf' '(' **DataRange** **DataRange** { **DataRange** } ')'

subClassExpression :=
    Class |
    subObjectSomeValuesFrom | **DataSomeValuesFrom**

subObjectSomeValuesFrom := 'ObjectSomeValuesFrom' '(' **ObjectPropertyExpression** *owl:Thing* ')'

superClassExpression :=
    Class |
    superObjectIntersectionOf | **superObjectComplementOf** |
    superObjectSomeValuesFrom | **DataSomeValuesFrom**

superObjectIntersectionOf := 'ObjectIntersectionOf' '(' **superClassExpression** **superClassExpression** { **superClassExpression** } ')'

superObjectComplementOf := 'ObjectComplementOf' '(' **subClassExpression** ')'

superObjectSomeValuesFrom := 'ObjectSomeValuesFrom' '(' **ObjectPropertyExpression** **Class** ')'

DataSomeValuesFrom := 'DataSomeValuesFrom' '(' **DataPropertyExpression** **DataRange** ')'

Axiom := **Declaration** | **ClassAxiom** | **ObjectPropertyAxiom** | **DataPropertyAxiom** | **DatatypeDefinition** | **Assertion** | **AnnotationAxiom**

ClassAxiom := **SubClassOf** | **EquivalentClasses** | **DisjointClasses**

SubClassOf := 'SubClassOf' '(' **axiomAnnotations** **subClassExpression** **superClassExpression** ')'

EquivalentClasses := 'EquivalentClasses' '(' **axiomAnnotations** **subClassExpression** **subClassExpression** { **subClassExpression** } ')'

DisjointClasses := 'DisjointClasses' '(' **axiomAnnotations** **subClassExpression** **subClassExpression** { **subClassExpression** } ')'

ObjectPropertyAxiom :=
SubObjectPropertyOf | **EquivalentObjectProperties** |
DisjointObjectProperties | **InverseObjectProperties** |
ObjectPropertyDomain | **ObjectPropertyRange** |
ReflexiveObjectProperty |
SymmetricObjectProperty | **AsymmetricObjectProperty**

SubObjectPropertyOf := 'SubObjectPropertyOf' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** ')'

EquivalentObjectProperties := 'EquivalentObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** { **ObjectPropertyExpression** } ')'

DisjointObjectProperties := 'DisjointObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** {

W3C Editor's Draft

ObjectPropertyExpression } ')'

InverseObjectProperties := 'InverseObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** ')'

ObjectPropertyDomain := 'ObjectPropertyDomain' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'

ObjectPropertyRange := 'ObjectPropertyRange' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'

ReflexiveObjectProperty := 'ReflexiveObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

SymmetricObjectProperty := 'SymmetricObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

AsymmetricObjectProperty := 'AsymmetricObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

DataPropertyAxiom :=
SubDataPropertyOf | **EquivalentDataProperties** | **DisjointDataProperties** |
DataPropertyDomain | **DataPropertyRange**

SubDataPropertyOf := 'SubDataPropertyOf' '(' **axiomAnnotations** **subDataPropertyExpression** **superDataPropertyExpression** ')'
subDataPropertyExpression := **DataPropertyExpression**
superDataPropertyExpression := **DataPropertyExpression**

EquivalentDataProperties := 'EquivalentDataProperties' '(' **axiomAnnotations** **DataPropertyExpression** **DataPropertyExpression** { **DataPropertyExpression** } ')'

DisjointDataProperties := 'DisjointDataProperties' '(' **axiomAnnotations** **DataPropertyExpression** **DataPropertyExpression** { **DataPropertyExpression** } ')'

DataPropertyDomain := 'DataPropertyDomain' '(' **axiomAnnotations** **DataPropertyExpression** **superClassExpression** ')'

DataPropertyRange := 'DataPropertyRange' '(' **axiomAnnotations** **DataPropertyExpression** **DataRange** ')'

DatatypeDefinition := 'DatatypeDefinition' '(' **axiomAnnotations** **Datatype** **DataRange** ')'

Assertion := **DifferentIndividuals** | **ClassAssertion** | **ObjectPropertyAssertion** | **DataPropertyAssertion**

sourceIndividual := **Individual**
targetIndividual := **Individual**
targetValue := **Literal**

DifferentIndividuals := 'DifferentIndividuals' '(' **axiomAnnotations** **Individual** **Individual** { **Individual** } ')'

ClassAssertion := 'ClassAssertion' '(' **axiomAnnotations** **Class** **Individual** ')'

ObjectPropertyAssertion := 'ObjectPropertyAssertion' '(' **axiomAnnotations** **ObjectPropertyExpression** **sourceIndividual** **targetIndividual** ')'

DataPropertyAssertion := 'DataPropertyAssertion' '(' **axiomAnnotations** **DataPropertyExpression** **sourceIndividual** **targetValue** ')'

6.3 OWL 2 RL

The grammar of OWL 2 RL consists of the general definitions from Section 13.1 of the OWL 2 Specification [*OWL 2 Specification*], as well as the following productions.

Class := **IRI**

Datatype := **IRI**

ObjectProperty := **IRI**

DataProperty := **IRI**

W3C Editor's Draft

```
AnnotationProperty := IRI

Individual := NamedIndividual | AnonymousIndividual

NamedIndividual := IRI

AnonymousIndividual := nodeID

Literal := typedLiteral | stringLiteralNoLanguage |
stringLiteralWithLanguage
typedLiteral := lexicalForm '^^' Datatype
lexicalForm := quotedString
stringLiteralNoLanguage := quotedString
stringLiteralWithLanguage := quotedString languageTag



ObjectPropertyExpression := ObjectProperty | InverseObjectProperty

InverseObjectProperty := 'ObjectInverseOf' '(' ObjectProperty ')'

DataPropertyExpression := DataProperty



DataRange := Datatype | DataIntersectionOf

DataIntersectionOf := 'DataIntersectionOf' '(' DataRange
DataRange { DataRange } ')'



zeroOrOne := '0' | '1'

subClassExpression :=
    Class other than owl:Thing |
    subObjectIntersectionOf | subObjectUnionOf | ObjectOneOf |
    subObjectSomeValuesFrom | ObjectHasValue |
    DataSomeValuesFrom | DataHasValue

subObjectIntersectionOf := 'ObjectIntersectionOf' '('
subClassExpression subClassExpression { subClassExpression } ')'

subObjectUnionOf := 'ObjectUnionOf' '(' subClassExpression
subClassExpression { subClassExpression } ')'
```

W3C Editor's Draft

```
subObjectSomeValuesFrom :=
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression
subClassExpression ')' |
    'ObjectSomeValuesFrom' '(' ObjectPropertyExpression
owl:Thing ')'

superClassExpression :=
    Class other than owl:Thing |
    superObjectIntersectionOf | superComplementOf |
    superObjectAllValuesFrom | ObjectHasValue |
superObjectMaxCardinality |
    DataAllValuesFrom | DataHasValue | superDataMaxCardinality

superObjectIntersectionOf := 'ObjectIntersectionOf' '('
superClassExpression superClassExpression { superClassExpression }
')'

superObjectComplementOf := 'ObjectComplementOf' '('
subClassExpression ')'

superObjectAllValuesFrom := 'ObjectAllValuesFrom' '('
ObjectPropertyExpression superClassExpression ')'

superObjectMaxCardinality :=
    'ObjectMaxCardinality' '(' zeroOrOne
ObjectPropertyExpression [ subClassExpression ] ')' |
    'ObjectMaxCardinality' '(' zeroOrOne
ObjectPropertyExpression owl:Thing ')'

superDataMaxCardinality := 'DataMaxCardinality' '(' zeroOrOne
DataPropertyExpression [ DataRange ] ')' |

equivClassExpression :=
    Class other than owl:Thing |
    equivObjectIntersectionOf |
    ObjectHasValue |
    DataHasValue

equivObjectIntersectionOf := 'ObjectIntersectionOf' '('
equivClassExpression equivClassExpression { equivClassExpression }
')'

ObjectOneOf := 'ObjectOneOf' '(' Individual { Individual }')'

ObjectHasValue := 'ObjectHasValue' '(' ObjectPropertyExpression
Individual ')'
```

W3C Editor's Draft

DataSomeValuesFrom := 'DataSomeValuesFrom' '(' **DataPropertyExpression** { **DataPropertyExpression** } **DataRange** ')'

DataAllValuesFrom := 'DataAllValuesFrom' '(' **DataPropertyExpression** { **DataPropertyExpression** } **DataRange** ')'

DataHasValue := 'DataHasValue' '(' **DataPropertyExpression** **Literal** ')'

Axiom := **Declaration** | **ClassAxiom** | **ObjectPropertyAxiom** | **DataPropertyAxiom** | **DatatypeDefinition** | **HasKey** | **Assertion** | **AnnotationAxiom**

ClassAxiom := **SubClassOf** | **EquivalentClasses** | **DisjointClasses**

SubClassOf := 'SubClassOf' '(' **axiomAnnotations** **subClassExpression** **superClassExpression** ')'

EquivalentClasses := 'EquivalentClasses' '(' **axiomAnnotations** **equivClassExpression** **equivClassExpression** { **equivClassExpression** } ')'

DisjointClasses := 'DisjointClasses' '(' **axiomAnnotations** **subClassExpression** **subClassExpression** { **subClassExpression** } ')'

ObjectPropertyAxiom :=
    SubObjectPropertyOf | **EquivalentObjectProperties** |
    DisjointObjectProperties | **InverseObjectProperties** |
    ObjectPropertyDomain | **ObjectPropertyRange** |
    FunctionalObjectProperty | **InverseFunctionalObjectProperty** |
    IrreflexiveObjectProperty |
    SymmetricObjectProperty | **AsymmetricObjectProperty**
    TransitiveObjectProperty

SubObjectPropertyOf := 'SubObjectPropertyOf' '(' **axiomAnnotations** **subObjectPropertyExpression** **superObjectPropertyExpression** ')'
subObjectPropertyExpression := **ObjectPropertyExpression** | **propertyExpressionChain**

propertyExpressionChain := 'ObjectPropertyChain' '(' **ObjectPropertyExpression** **ObjectPropertyExpression** { **ObjectPropertyExpression** } ')'
superObjectPropertyExpression := **ObjectPropertyExpression**

EquivalentObjectProperties := 'EquivalentObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** { **ObjectPropertyExpression** } ')'

DisjointObjectProperties := 'DisjointObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** { **ObjectPropertyExpression** } ')'

InverseObjectProperties := 'InverseObjectProperties' '(' **axiomAnnotations** **ObjectPropertyExpression** **ObjectPropertyExpression** ')'

ObjectPropertyDomain := 'ObjectPropertyDomain' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'

ObjectPropertyRange := 'ObjectPropertyRange' '(' **axiomAnnotations** **ObjectPropertyExpression** **superClassExpression** ')'

FunctionalObjectProperty := 'FunctionalObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

InverseFunctionalObjectProperty := 'InverseFunctionalObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

ReflexiveObjectProperty := 'ReflexiveObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

IrreflexiveObjectProperty := 'IrreflexiveObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

SymmetricObjectProperty := 'SymmetricObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

AsymmetricObjectProperty := 'AsymmetricObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

TransitiveObjectProperty := 'TransitiveObjectProperty' '(' **axiomAnnotations** **ObjectPropertyExpression** ')'

```
DataPropertyAxiom :=
    SubDataPropertyOf | EquivalentDataProperties |
DisjointDataProperties |
    DataPropertyDomain | DataPropertyRange | FunctionalDataProperty

SubDataPropertyOf := 'SubDataPropertyOf' '(' axiomAnnotations
subDataPropertyExpression superDataPropertyExpression ')'
subDataPropertyExpression := DataPropertyExpression
superDataPropertyExpression := DataPropertyExpression

EquivalentDataProperties := 'EquivalentDataProperties' '('
axiomAnnotations DataPropertyExpression DataPropertyExpression {
DataPropertyExpression } ')'

DisjointDataProperties := 'DisjointDataProperties' '('
axiomAnnotations DataPropertyExpression DataPropertyExpression {
DataPropertyExpression } ')'

DataPropertyDomain := 'DataPropertyDomain' '(' axiomAnnotations
DataPropertyExpression superClassExpression ')'

DataPropertyRange := 'DataPropertyRange' '(' axiomAnnotations
DataPropertyExpression DataRange ')'

FunctionalDataProperty := 'FunctionalDataProperty' '('
axiomAnnotations DataPropertyExpression ')'


DatatypeDefinition := 'DatatypeDefinition' '(' axiomAnnotations
Datatype DataRange ')'


HasKey := 'HasKey' '(' axiomAnnotations subClassExpression '(' {
ObjectPropertyExpression } ')' '(' { DataPropertyExpression } ')'
')'


Assertion :=
    SameIndividual | DifferentIndividuals | ClassAssertion |
    ObjectPropertyAssertion | NegativeObjectPropertyAssertion |
    DataPropertyAssertion | NegativeDataPropertyAssertion
```

```
sourceIndividual := Individual
targetIndividual := Individual
targetValue := Literal

SameIndividual := 'SameIndividual' '(' axiomAnnotations Individual
Individual { Individual } ')'

DifferentIndividuals := 'DifferentIndividuals' '(' axiomAnnotations
Individual Individual { Individual } ')'

ClassAssertion := 'ClassAssertion' '(' axiomAnnotations
superClassExpression Individual ')'

ObjectPropertyAssertion := 'ObjectPropertyAssertion' '('
axiomAnnotations ObjectPropertyExpression sourceIndividual
targetIndividual ')'

NegativeObjectPropertyAssertion :=
'NegativeObjectPropertyAssertion' '(' axiomAnnotations
ObjectPropertyExpression sourceIndividual targetIndividual ')'

DataPropertyAssertion := 'DataPropertyAssertion' '('
axiomAnnotations DataPropertyExpression sourceIndividual targetValue
')'

NegativeDataPropertyAssertion := 'NegativeDataPropertyAssertion'
'(' axiomAnnotations DataPropertyExpression sourceIndividual
targetValue ')'
```

7 Acknowledgments

The starting point for the development of OWL 2 was the OWL1.1 member submission, itself a result of user and developer feedback, and in particular of information gathered during the OWL Experiences and Directions (OWLED) Workshop series. The working group also considered postponed issues from the WebOnt Working Group.

This document has been produced by the OWL Working Group (see below), and its contents reflect extensive discussions within the Working Group as a whole. The editors extend special thanks to Jie Bao (RPI), Jim Hendler (RPI) and Jeff Pan (University of Aberdeen) for their thorough reviews.

The regular attendees at meetings of the OWL Working Group at the time of publication of this document were: Jie Bao (RPI), Diego Calvanese (Free University of Bozen-Bolzano), Bernardo Cuenca Grau (Oxford University), Martin Dzbor

W3C Editor's Draft

(Open University), Achille Fokoue (IBM Corporation), Christine Golbreich (Université de Versailles St-Quentin and LIRMM), Sandro Hawke (W3C/MIT), Ivan Herman (W3C/ERCIM), Rinke Hoekstra (University of Amsterdam), Ian Horrocks (Oxford University), Elisa Kendall (Sandpiper Software), Markus Krötzsch (FZI), Carsten Lutz (Universität Bremen), Deborah L. McGuinness (RPI), Boris Motik (Oxford University), Jeff Pan (University of Aberdeen), Bijan Parsia (University of Manchester), Peter F. Patel-Schneider (Bell Labs Research, Alcatel-Lucent), Alan Ruttenberg (Science Commons), Uli Sattler (University of Manchester), Michael Schneider (FZI), Mike Smith (Clark & Parsia), Evan Wallace (NIST), and Zhe Wu (Oracle Corporation). We would also like to thank past members of the working group: Jeremy Carroll, Jim Hendler, Vipul Kashyap.

8 References

8.1 Normative References

[OWL 2 Conformance]
OWL 2 Web Ontology Language: Conformance Michael Smith, Ian Horrocks, Markus Krötzsch, eds. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-test-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-test/.

[OWL 2 Direct Semantics]
OWL 2 Web Ontology Language: Direct Semantics Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-semantics-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-semantics/.

[OWL 2 RDF Mapping]
OWL 2 Web Ontology Language: Mapping to RDF Graphs Peter F. Patel-Schneider, Boris Motik, eds. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-mapping-to-rdf-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-mapping-to-rdf/.

[OWL 2 RDF-Based Semantics]
OWL 2 Web Ontology Language: RDF-Based Semantics Michael Schneider, editor. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-rdf-based-semantics-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-rdf-based-semantics/.

[OWL 2 Specification]
OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax Boris Motik, Peter F. Patel-Schneider, Bijan Parsia, eds. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-syntax-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-syntax/.

[RFC 2119]
RFC 2119: Key words for use in RFCs to Indicate Requirement Levels. Network Working Group, S. Bradner. IETF, March 1997, http://www.ietf.org/rfc/rfc2119.txt

W3C Editor's Draft

8.2 Nonnormative References

[Complexity]
Complexity Results and Practical Algorithms for Logics in Knowledge Representation. Stephan Tobies. Ph.D Dissertation, 2002

[DL-Lite]
Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family. Diego Calvanese, Giuseppe de Giacomo, Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati. J. of Automated Reasoning 39(3):385–429, 2007

[DL-Lite-bool]
The DL-Lite Family and Relatives. Alessandro Artale, Diego Calvanese, Roman Kontchakov, Michael Zakharyaschev, submitted.

[DLP]
Description Logic Programs: Combining Logic Programs with Description Logic. Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. in Proc. of the 12th Int. World Wide Web Conference (WWW 2003), Budapest, Hungary, 2003. pp.: 48–57

[EL++]
Pushing the EL Envelope. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the 19th Joint Int. Conf. on Artificial Intelligence (IJCAI 2005), 2005

[EL++ Update]
Pushing the EL Envelope Further. Franz Baader, Sebastian Brandt, and Carsten Lutz. In Proc. of the Washington DC workshop on OWL: Experiences and Directions (OWLED08DC), 2008

[OWL 1 Reference]
OWL Web Ontology Language Reference. Mike Dean and Guus Screiber, eds., W3C Recommendation, 10 February 2004.

[OWL 2 Primer]
OWL 2 Web Ontology Language: Primer Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph. W3C Editor's Draft, 21 May 2009, http://www.w3.org/2007/OWL/draft/ED-owl2-primer-20090521/. Latest version available at http://www.w3.org/2007/OWL/draft/owl2-primer/.

[Papadimitriou]
Christos H. Papadimitriou. Computational Complexity. Addison Wesley Publ. Co., Reading, Massachussetts, 1994.

[pD*]
Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. Herman J. ter Horst. J. of Web Semantics 3(2–3):79–115, 2005

[RDF Semantics]
RDF Semantics. Patrick Hayes, ed., W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available as http://www.w3.org/TR/rdf-mt/.

[RIF RDF & OWL]
RIF RDF and OWL Compatibility. Jos de Bruijn, ed., W3C Working Draft, 30 July 2008.

[SNOMED CT]
: *Systematized Nomenclature of Medicine – Clinical Terms*. International Health Terminology Standards Development Organization (IHTSDO)

[Automated Reasoning]
: *Handbook of Automated Reasoning*. A. Robinson and A. Voronkov, eds., Elsevier Science, 2001

[Logic Programming]
: *Logic programming and knowledge representation*. Chitta Baral and Michael Gelfond. Journal of Logic Programming 19:73–148, 1994.