Validating RDF with OWL Integrity Constraints

Authors:
Héctor Pérez-Urbina, Clark & Parsia, LLC
Evren Sirin, Clark & Parsia, LLC
Kendall Clark, Clark & Parsia, LLC

http://docs.stardog.com/icv/icv-specification.html

Abstract

This document proposes a method for validating Semantic Web and Linked Data by providing an alternative, integrity constraint (IC) semantics for OWL. A model-theoretic semantics based on the Closed World Assumption and a weak variant of the Unique Name Assumption is given for OWL axioms that are thereby interpreted as ICs. The document includes a structural specification in order to augment an ontology with a set of OWL ICs and a brief description of possible implementation approaches.

Table of Contents

1. Introduction

This document proposes a method of constraining and validating Semantic Web and Linked Data (i.e., RDF) instance data using Integrity Constraints (ICs) modeled as OWL axioms. The proposal enables an OWL ontology to be interpreted as a set of ICs; i.e., checks that must be satisfied by the information explicitly present or information that may be inferred. We define a model-theoretic semantics for OWL ICs based on the Closed World Assumption and a weak variant of the Unique Name Assumption and briefly describe feasible implementation strategies.

In some use cases and for some requirements, OWL users intend OWL axioms to be interpreted as ICs. However, the direct semantics of OWL [*OWL 2 Direct Semantics*] does not interpret OWL axioms in this way; thus, the consequences that one can draw from such ontologies differ from the ones that some users intuitively expect and require. In other words, some users want to use OWL as a validation or constraint language for RDF instance data, but that is not possible using OWL software that correctly implements the existing semantics of OWL. This document addresses that situation by providing a different semantics—compatible with the existing semantics—for OWL axioms which may be used, together with appropriate software, to validate RDF instance data.

To see the nature of the problem, consider an OWL ontology that describes terms and concepts regarding the product inventory of a supermarket. The ontology includes the classes *Product* and *Provider*, the object property *hasProvider*, and the data property *hasID*.

Suppose we want to impose the following ICs on the data:

I. Each product must have an ID
II. Only products can have IDs
III. Products must not have more than one provider

These constraints could be interpreted in the following way:

I. Whenever an instance $product_i$ of *Product* is added to the ontology, a check should be performed to verify whether the ID of $product_i$ has been specified; if not, the update should be rejected.
II. Whenever an instance $<product_i, ID_i>$ of *hasID* is added to the ontology, a check should be performed to verify whether $product_i$ is an instance of *Product*; if not, the update should be rejected.
III. Whenever an instance $<product_i, provider_i>$ of *hasProvider* is added to the ontology, a check should be performed to verify whether another provider $provider_j$ has been specified for $product_i$; if so, the update should be rejected.

These constraints can be concisely and unambiguously represented as OWL axioms:

I. Class: Product
hasID some literal

II. DataProperty: hasID
Domain: Product

III. ObjectProperty: hasProvider
Characteristics: Functional

However, these axioms will not be interpreted as checks by software which implements the standard OWL semantics. In fact, according to the standard OWL semantics, we have that:

I. Adding a product without an ID to the ontology does not raise an error, but leads to the inference that the product in question has an unknown ID.
II. Adding a tuple $<product_i, ID_i>$ to the ontology without $product_i$ being an instance of *Product* does not raise an error, but leads to the inference that $product_i$ is an instance of *Product*.
III. Adding a tuple $<product_i, provider_i>$ having specified a previous provider $provider_j$ for $product_i$ does not raise an error, but leads to the inference that $provider_i$ and $provider_j$ denote the same individual.

In some cases, users want these inferences; but in others, users want integrity constraint violations to be detected, reported, repaired, etc.

OWL adopts the Open World Assumption (OWA) and does not adopt the Unique Name Assumption (UNA). These design choices make it very difficult to treat these axioms as ICs. On the one hand, due to OWA, a statement must not be inferred to be false on the basis of failures to prove it; therefore, the fact that a piece of information has not been specified (e.g., a product's ID) does not mean that such information does not exist. On the other hand, the absence of UNA allows two different constants to refer to the same individual (e.g., $provider_i$ and $provider_j$).

The standard interpretation of OWL axioms that are intended to be interpreted as ICs is inappropriate for some use cases and applications; therefore, it is useful to define an alternate semantics for OWL based on IC. An IC semantics together with associated software will increase the number of satisfied users of OWL because OWL and the software will then behave as those users intuitively expect and require.

As formally defined in Section 2, our approach allows for a standard OWL ontology O to import a set of IC ontologies—OWL ontologies that are to be interpreted as ICs. Note that the IC semantics for OWL defined in this document is a strict extension of the standard OWL semantics: in case O imports no IC ontology, then O should be interpreted as a standard OWL ontology.

2. Structural Specification

An OWL ontology that is to be interpreted as a set of ICs is called an *IC ontology*. We slightly extend the structural specification of OWL in order to allow ontologies to import a set of IC ontologies. We do so by introducing a new annotation property which is defined analogously to owl:imports—the annotation property that is used to import standard ontologies defined by OWL 2 [*OWL 2 Specification*].

We use an annotation property that resembles owl:imports with a different namespace: http://www.w3.org/Submission/owlic/. In the following, we denote this annotation property as ic:imports for brevity.

An example usage of this annotation property is given in the following:

```
Namespace(ic = <http://www.w3.org/Submission/owlic/>)
Ontology(<http://www.example.com/instanceOntology>
        Import(<http://www.example.com/schemaOntology>)
        Annotation(ic:imports <http://www.example.com/constraintsOntology>)
        ...
)
```

where instanceOntology imports the standard axioms of schemaOntology and imports constraintsOntology as an IC ontology. This import approach to relating ICs to other OWL ontologies gives enough flexibility to users without too much maintenance cost and negligible impact on existing tools. See Implementation Remarks for a more detailed discussion of this design choice.

An OWL ontology can import a set of IC ontologies via ic:imports. An IC ontology can import a set of IC ontologies via ic:imports as well. And, of course, an IC ontology can import a set of standard ontologies via owl:imports as usual.

The *import closure* of an IC ontology is defined in the same vein as the import closure for standard OWL ontologies. The *IC import closure* of a standard or IC ontology O is a set containing all the IC ontologies that O imports via the ic:imports annotation property. The *IC closure* of a standard or IC ontology O is the smallest set that contains all the axioms from each ontology O' in the IC import closure of O.

3. OWL IC Semantics

We refer to the definitions of datatype map, vocabulary, and OWL interpretation and model in OWL 2 [*OWL 2 Direct Semantics*].

3.1 IC-Interpretations

Let $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$ be a datatype map and let $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ be a vocabulary over D. An *IC-interpretation* $\mathcal{I} = (\Delta_I, \Delta_D, I, U, \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for D and V is an 11-tuple with the following structure:

- Δ_I is a nonempty set called the *object domain*.
- Δ_D is a nonempty set disjoint with Δ_I called the *data domain* such that $(DT)^{DT} \subseteq \Delta_D$ for

each datatype $DT \in V_{DT}$.

- $I = (\Delta_I, \Delta_D, \cdot^{C_I}, \cdot^{OP_I}, \cdot^{DP_I}, \cdot^{I}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ is an OWL interpretation for D and V.
- $U = \{U_1, U_2, ..., U_n\}$ is a set where each $U_j = (\Delta_{I_j}, \Delta_{D_j}, \cdot^{C_j}, \cdot^{OP_j}, \cdot^{DP_j}, \cdot^{I_j}, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for $1 \leq j \leq n$ is an OWL interpretation for D and V.
- $\cdot^{C}$ is the *class interpretation function* that assigns to each class $C \in V_C$ a subset $(C)^C \subseteq \Delta_I$ such that
  - $(owl:Thing)^C = \Delta_I$,
  - $(owl:Nothing)^C = \varnothing$, and
  - $(C)^C = \{x^I \mid x \in V_I$ and for each $U_j \in U$ we have that $x^{I_{U_j}} \in (C)^{C_j}\}$.
- $\cdot^{OP}$ is the *object property interpretation function* that assigns to each object property $OP \in V_{OP}$ a subset $(OP)^{OP} \subseteq \Delta_I \times \Delta_I$ such that
  - $(owl:topObjectProperty)^{OP} = \Delta_I \times \Delta_I$,
  - $(owl:bottomObjectProperty)^{OP} = \varnothing$, and
  - $(OP)^{OP} = \{(x^I, y^I) \mid x \in V_I$, y $\in$ V$_I$, and for each $U_j \in U$ we have that $(x^{I_{U_j}}, y^{I_{U_j}}) \in (OP)^{OP_j}\}$.
- $\cdot^{DP}$ is the *data property interpretation function* that assigns to each data property $DP \in V_{DP}$ a subset $(DP)^{DP} \subseteq \Delta_I \times \Delta_D$ such that
  - $(owl:topDataProperty)^{DP} = \Delta_I \times \Delta_D$,
  - $(owl:bottomDataProperty)^{DP} = \varnothing$, and
  - $(DP)^{DP} = \{(x^I, lt^{LT}) \mid x \in V_I, lt \in V_{LT}$, and for each $U_j \in U$ we have that $(x^{I_{U_j}}, lt^{LT}) \in (DP)^{DP_j}\}$.
- $\cdot^{I}$ is the *individual interpretation function* that assigns to each individual $a \in V_I$ an element $(a)^I \in \Delta_I$.
- $\cdot^{DT}$ is the *datatype interpretation function* that assigns to each datatype $DT \in V_{DT}$ a subset $(DT)^{DT} \subseteq \Delta_D$ such that
  - $\cdot^{DT}$ is the same as in D for each datatype $DT \in N_{DT}$, and
  - $(rdfs:Literal)^{DT} = \Delta_D$.
- $\cdot^{LT}$ is the *literal interpretation function* that is defined as $(lt)^{LT} = (LV, DT)^{LS}$ for each $lt \in V_{LT}$, where LV is the lexical form of lt and DT is the datatype of lt.
- $\cdot^{FA}$ is the *facet interpretation function* that is defined as $(F, lt)^{FA} = (F, (lt)^{LT})^{FS}$ for each $(F, lt) \in V_{FA}$.

The extensions of $\cdot^{C}$, $\cdot^{OP}$, and $\cdot^{DT}$ to class expressions, object property expressions, and data ranges respectively, are defined analogously to OWL 2 [OWL 2 Direct Semantics]. For example, we extend $\cdot^{C}$ to the class expression $ObjectIntersectionOf(CE_1 ... CE_n)$ as $(CE_1)^C \cap ... \cap (CE_n)^C$. However, we extend $\cdot^{C}$ to $ObjectComplementOf(CE)$ as $\{x^I \mid x \in V_I\} \setminus (CE)^C$—that is, the complement of a class expression is defined with respect to the set of *named individuals* as opposed to the object domain. The complete extensions for $\cdot^{C}$, $\cdot^{OP}$, and $\cdot^{DT}$ can be found in the Appendix.

3.2 Axiom Satisfaction

Satisfaction of an IC-interpretation $\mathcal{I}$ with respect to a given axiom is defined analogously to satisfaction of standard intepretations defined in OWL 2 [OWL 2 Direct Semantics]. For example, $\mathcal{I}$ satisfies the axiom $SubClassOf(CE_1\ CE_2)$ if $(CE_1)^C \subseteq (CE_2)^C$. The complete definitions for axiom satisfaction can be found in the Appendix.

3.3 Axiom IC-Satisfaction

Let $D = (N_{DT}, N_{LS}, N_{FS}, \cdot^{DT}, \cdot^{LS}, \cdot^{FS})$ be a datatype map and let $V = (V_C, V_{OP}, V_{DP}, V_I, V_{DT}, V_{LT}, V_{FA})$ be a vocabulary over D.

- Let I be an OWL interpretation for D and V. With E_I we denote the set of equality relations between named individuals satisfied by I. That is, $E_I = \{ \langle a, b \rangle \mid a \in V_I, b \in V_I,$ and $a^I = b^I \}$ where $\cdot^I$ is the individual interpretation function of I.
- Let I and J be OWL interpretations for D and V. We say that $I <_{eq} J$ iff the following conditions are satisfied:
  - for every $C \in V_C$, I satisfies $C(a)$ iff J satisfies $C(a)$
  - for every $R \in V_{OP}$, I satisfies $R(a, b)$ iff J satisfies $R(a, b)$
  - for every $S \in V_{DP}$, I satisfies $S(a, lt)$ iff J satisfies $S(a, lt)$
  - $E_I \subset E_J$
- Let O be an OWL ontology. With $Mod(O)$ we denote the set containing exactly all the models of O with respect to D and V. With $Mod_{ME}(O)$ we denote the set containing exactly all the models with minimal equality between named individuals. That is, $Mod_{ME}(O) = \{ I \mid I \in Mod(O)$ and there is no J such that $J \in Mod(O)$ and $J <_{eq} I \}$.
- Let O be an OWL ontology and α be an axiom. We say that O *IC-satisfies* α iff for all $I \in Mod_{ME}(O)$, we have that the IC-interpretation $\Gamma = (\Delta_I, \Delta_D, I, Mod_{ME}(O), \cdot^C, \cdot^{OP}, \cdot^{DP}, \cdot^I, \cdot^{DT}, \cdot^{LT}, \cdot^{FA})$ for D and V satisfies α.

3.4 Inference Problem

We are mainly interested in the following inference problem:

Ontology Validation. Let O be an OWL ontology. We say that O is *Valid* iff for all axioms α in the IC closure of O, it holds that O *IC-satisfies* α.

4. Implementation Remarks

4.1 Implementing IC Syntax

As discussed in Section 2, we use standard OWL syntax for ICs; store ICs in a separate document; and define a new annotation property analogous to owl:imports, that will associate a standard OWL ontology with a set of ICs defined for that ontology.

The motivation for this design choice is to minimize the effects of ICs on existing tools. From the perspective of creation and maintenance, users can continue using existing ontology authoring toolsets. For example, one can use an OWL editor to create ICs and store them in a document. The ontology for which the constraints are written can be augmented with the IC import annotation easily, since this is a standard OWL annotation. With an OWL editor that allows users to open and edit multiple ontologies at the same time, the regular ontology and the IC ontology can be edited together. Several OWL editors provide the feature to move axioms between ontologies; hence, one can easily change the interpretation of an axiom just by moving it from the regular ontology to the IC ontology.

The only issue in using an existing OWL editor is as follows: when a user opens a regular ontology that links to an IC ontology, the editor will not open the IC ontology automatically. The user needs to look at the ontology annotation and open the IC ontology manually. However, this is not a serious issue since it is a relatively simple extension for editors to recognize this annotation. It is safe to assume that such extensions will be available, especially if ICs start to be widely used.

Our approach has no impact on OWL existing reasoners that do not support ICs: since the

annotation property has no semantic effect, they would not process that annotation. Therefore, there is no additional work that needs to be done to hide the ICs in order to avoid unintended inferences that would occur if they are inadvertently interpreted as regular OWL axioms.

Note that our approach does not require an IC ontology to be identified as such. However, in case an ontology is to be exclusively interpreted as a set of ICs, one might use an ontology annotation to make this fact explicit. As usual, such annotations are for informational purposes only and have no effect on the semantics.

4.2 Implementing IC Semantics

The IC semantics described in this document is strongly related to the semantics presented in a paper [*TSBM10*] giving a formal integrity constraint semantics for the description logic SROIQ. Based on the correspondence between SROIQ and OWL 2 semantics [*OWL 2 Direct Semantics*], the semantics we present here has been adapted to OWL 2 and extended to support datatypes.

As discussed in the paper, there is a close relationship between IC semantics and queries that have negation as failure (NAF) operator. This is interesting from a practical point of view because a validator for OWL IC can be implemented in a straightforward way: each axiom in an IC ontology defined with respect to an OWL ontology *O* can be effectively transformed into a SPARQL query that can be later answered over *O* using the SPARQL entailment regime that corresponds to *O*.

As a simple example of the translation, consider the IC presented in Example 6 above:

```
Class: Supervisor
  SubClassOf: supervises some Employee
```

The translation of this IC to SPARQL would yield the following SPARQL query:

```
ASK WHERE {
  ?x rdf:type :Supervisor .
  OPTIONAL {
    ?x :supervises ?y .
    ?y rdf:type :Employee .
  }
  FILTER ( !bound( ?y ) )
}
```

If the execution of the query over an ontology *O* returns true, we can conclude that the IC has been violated by *O*; and, therefore, that *O* is *not* IC-valid with respect to this constraint. Note that the query uses the OPTIONAL/FILTER/!BOUND pattern to encode NAF. However, it is likely that SPARQL 1.1 [*SPARQL 1.1*] will make NAF more clearly visible syntatically, perhaps via NOT EXISTS as in current drafts.

It has been shown that SPARQL [*SPARQL*] has the same expressive power as nonrecursive datalog programs with NAF [*AG08*]. Therefore, it is possible to translate OWL ICs to a set of rules that will be evaluated over an ontology *O*. Such rules can be written using RIF Framework for Logic Dialects [*RIF-FLD*] with the Naf operator:

```
Forall ?x ?y (
   invalid() :- And (
      ?x[rdf:type -> :Supervisor]
      Naf And (
         ?x[:supervises -> ?y]
         ?y[rdf:type -> :Employee] )))
```

This rule uses the Naf operator for encoding NAF and defines an arbitrary RIF predicate

invalid to detect the condition that an ontology *O* is invalid with respect to ICs. Implementations would be free to choose a different name for the predicate.

Details of the translation are out of the scope here; interested readers are referred to the formal semantics paper mentioned previously [*TSBM10*]. Translation-based IC validation is one of many possibilities to implement IC validation and has been mentioned here as an example. An IC validator conforming to the IC semantics described here can also be implemented with different approaches.

Acknowledgements

We wish to thank the following people for their assistance: Pavel Klinov, Michael Smith, Michael Grove, Jiao Tao, and Peter Patel-Schneider. We thank members of the OWLED community, including the anonymous reviewers, who gave us very early feedback on using OWL as integrity constraints, including, most helpfully, use cases and requirements. We also acknowledge the support of NIST SBIR funding under the auspices of which this document was prepared.

References

[OWL 2 Direct Semantics]
OWL 2 Web Ontology Language: Direct Semantics. Boris Motik, Peter F. Patel-Schneider, Bernardo Cuenca Grau, eds. W3C Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-direct-semantics-20091027/. Latest version available at http://www.w3.org/TR/owl2-direct-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 Recommendation, 27 October 2009, http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/. Latest version available at http://www.w3.org/TR/owl2-syntax/.

[SPARQL]
SPARQL Query Language for RDF. Eric Prud'hommeaux and Andy Seaborne, eds. W3C Recommendation, 15 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Latest version available as http://www.w3.org/TR/rdf-sparql-query/.

[SPARQL-1.1]
SPARQL Query Language 1.1. Steve Harris and Andy Seaborne, eds. W3C Working Draft, 26 January 2010, http://www.w3.org/TR/2010/WD-sparql11-query-20100126/. Latest version available as http://www.w3.org/TR/sparql11-query/.

[TSBM10]
Integrity Constraints in OWL. Jiao Tao, Evren Sirin, Jie Bao, Deborah L. McGuinness. In Proc. of the 24th Conference on Artificial Intelligence (AAAI 2010). Atlanta, USA. 2010.

[AG08]
The Expressive Power of SPARQL. Renzo Angles, Claudio Gutierrez. In Proc. of the 7th International Semantic Web Conference (ISWC 2008). Karlsruhe, Germany. 2008.

Appendix A

A.1 Extensions of Interpretation Functions

A.1.1 Class Expressions

The class interpretation function $\cdot^{\mathcal{C}}$ is extended to class expressions as shown in Table 1. For *S* a set, *#S* denotes the number of elements in *S*.

Table 1. Interpreting Class Expressions

Class Expression	Interpretation $\cdot^C$
ObjectIntersectionOf(CE_1 ... CE_n)	$(CE_1)^C \cap ... \cap (CE_n)^C$
ObjectUnionOf(CE_1 ... CE_n)	$(CE_1)^C \cup ... \cup (CE_n)^C$
ObjectComplementOf(CE)	$\{ x^I \mid x \in V_I \} \setminus (CE)^C$
ObjectOneOf(a_1 ... a_n)	$\{ (a_1)^I, ..., (a_n)^I \}$
ObjectSomeValuesFrom(OPE CE)	$\{ x \mid \exists y: (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \}$
ObjectAllValuesFrom(OPE CE)	$\{ x \mid \forall y: (x, y) \in (OPE)^{OP}$ implies $y \in (CE)^C \}$
ObjectHasValue(OPE a)	$\{ x \mid (x, (a)^I) \in (OPE)^{OP} \}$
ObjectHasSelf(OPE)	$\{ x \mid (x, x) \in (OPE)^{OP} \}$
ObjectMinCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} \geq n \}$
ObjectMaxCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} \leq n \}$
ObjectExactCardinality(n OPE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP} \} = n \}$
ObjectMinCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} \geq n \}$
ObjectMaxCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} \leq n \}$
ObjectExactCardinality(n OPE CE)	$\{ x \mid \#\{ y \mid (x, y) \in (OPE)^{OP}$ and $y \in (CE)^C \} = n \}$
DataSomeValuesFrom(DPE_1 ... DPE_n DR)	$\{ x \mid \exists y_1, ..., y_n: (x, y_k) \in (DPE_k)^{DP}$ for each $1 \leq k \leq n$ and $(y_1, ..., y_n) \in (DR)^{DT} \}$
DataAllValuesFrom(DPE_1 ... DPE_n DR)	$\{ x \mid \forall y_1, ..., y_n: (x, y_k) \in (DPE_k)^{DP}$ for each $1 \leq k \leq n$ imply $(y_1, ..., y_n) \in (DR)^{DT} \}$
DataHasValue(DPE lt)	$\{ x \mid (x, (lt)^{LT}) \in (DPE)^{DP} \}$
DataMinCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP} \} \geq n \}$
DataMaxCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP} \} \leq n \}$
DataExactCardinality(n DPE)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP} \} = n \}$
DataMinCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} \geq n \}$
DataMaxCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} \leq n \}$
DataExactCardinality(n DPE DR)	$\{ x \mid \#\{ y \mid (x, y) \in (DPE)^{DP}$ and $y \in (DR)^{DT} \} = n \}$

A.1.2 Object Property Expressions

The object property interpretation function $\cdot^{OP}$ is extended to object property expressions as shown in Table 2.

Table 2. Interpreting Object Property Expressions

Object Property Expression	Interpretation $\cdot^{OP}$
ObjectInverseOf(OP)	$\{ (x, y) \mid (y, x) \in (OP)^{OP} \}$

A.1.3 Data Ranges

The datatype interpretation function $\cdot^{DT}$ is extended to data ranges as shown in Table 3. All datatypes in OWL 2 are unary, so each datatype DT is interpreted as a unary relation over Δ_D — that is, as a set $(DT)^{DT} \subseteq \Delta_D$. OWL 2 currently does not define data ranges of arity more than one; however, by allowing for n-ary data ranges, the syntax of OWL 2 provides a "hook" allowing implementations to introduce extensions such as comparisons and arithmetic. An n-ary data range DR is interpreted as an n-ary relation $(DR)^{DT}$ over Δ_D — that is, as a set $(DT)^{DT} \subseteq (\Delta_D)^n$.

Table 3. Interpreting Data Ranges

Data Range	Interpretation $\cdot^{DT}$
DataIntersectionOf(DR_1 ... DR_n)	$(DR_1)^{DT} \cap ... \cap (DR_n)^{DT}$
DataUnionOf(DR_1 ... DR_n)	$(DR_1)^{DT} \cup ... \cup (DR_n)^{DT}$
DataComplementOf(DR)	$(\Delta_D)^n \setminus (DR)^{DT}$ where n is the arity of DR
DataOneOf(lt_1 ... lt_n)	$\{ (lt_1)^{LT}, ..., (lt_n)^{LT} \}$
DatatypeRestriction(DT F_1 lt_1 ... F_n lt_n)	$(DT)^{DT} \cap (F_1, lt_1)^{FA} \cap ... \cap (F_n, lt_n)^{FA}$

A.2 Satisfaction of Axioms

A.2.1 Class Expression Axioms

Satisfaction of OWL 2 class expression axioms in Γ with respect to an ontology O is defined as shown in Table 4.

Table 4. Satisfaction of Class Expression Axioms in an Interpretation

Axiom	Condition
SubClassOf(CE_1 CE_2)	$(CE_1)^C \subseteq (CE_2)^C$
EquivalentClasses(CE_1 ... CE_n)	$(CE_j)^C = (CE_k)^C$ for each $1 \le j \le n$ and each $1 \le k \le n$
DisjointClasses(CE_1 ... CE_n)	$(CE_j)^C \cap (CE_k)^C = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
DisjointUnion(C CE_1 ... CE_n)	$(C)^C = (CE_1)^C \cup ... \cup (CE_n)^C$ and $(CE_j)^C \cap (CE_k)^C = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$

A.2.2 Object Property Expression Axioms

Satisfaction of OWL 2 object property expression axioms in Γ with respect to an ontology O is defined as shown in Table 5.

Table 5. Satisfaction of Object Property Expression Axioms in an Interpretation

Axiom	Condition
SubObjectPropertyOf(OPE_1 OPE_2)	$(OPE_1)^{OP} \subseteq (OPE_2)^{OP}$
SubObjectPropertyOf(ObjectPropertyChain(OPE_1 ... OPE_n) OPE)	$\forall y_0, ..., y_n$: (y_0, y_1) $\in (OPE_1)^{OP}$ and ... and (y_{n-1}, y_n) $\in (OPE_n)^{OP}$ imply (y_0, y_n) $\in (OPE)^{OP}$

EquivalentObjectProperties(OPE_1 ... OPE_n)	$(OPE_j)^{OP} = (OPE_k)^{OP}$ for each $1 \le j \le n$ and each $1 \le k \le n$
DisjointObjectProperties(OPE_1 ... OPE_n)	$(OPE_j)^{OP} \cap (OPE_k)^{OP} = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
ObjectPropertyDomain(OPE CE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $x \in (CE)^{C}$
ObjectPropertyRange(OPE CE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $y \in (CE)^{C}$
InverseObjectProperties(OPE_1 OPE_2)	$(OPE_1)^{OP} = \{ (x, y) \mid (y, x) \in (OPE_2)^{OP} \}$
FunctionalObjectProperty(OPE)	$\forall\, x, y_1, y_2$: $(x, y_1) \in (OPE)^{OP}$ and $(x, y_2) \in (OPE)^{OP}$ imply $y_1 = y_2$
InverseFunctionalObjectProperty(OPE)	$\forall\, x_1, x_2, y$: $(x_1, y) \in (OPE)^{OP}$ and $(x_2, y) \in (OPE)^{OP}$ imply $x_1 = x_2$
ReflexiveObjectProperty(OPE)	$\forall\, x$: $x \in \Delta_I$ implies $(x, x) \in (OPE)^{OP}$
IrreflexiveObjectProperty(OPE)	$\forall\, x$: $x \in \Delta_I$ implies $(x, x) \notin (OPE)^{OP}$
SymmetricObjectProperty(OPE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $(y, x) \in (OPE)^{OP}$
AsymmetricObjectProperty(OPE)	$\forall\, x, y$: $(x, y) \in (OPE)^{OP}$ implies $(y, x) \notin (OPE)^{OP}$
TransitiveObjectProperty(OPE)	$\forall\, x, y, z$: $(x, y) \in (OPE)^{OP}$ and $(y, z) \in (OPE)^{OP}$ imply $(x, z) \in (OPE)^{OP}$

A.2.3 Data Property Expression Axioms

Satisfaction of OWL 2 data property expression axioms in I with respect to an ontology O is defined as shown in Table 6.

Table 6. Satisfaction of Data Property Expression Axioms in an Interpretation

Axiom	Condition
SubDataPropertyOf(DPE_1 DPE_2)	$(DPE_1)^{DP} \subseteq (DPE_2)^{DP}$
EquivalentDataProperties(DPE_1 ... DPE_n)	$(DPE_j)^{DP} = (DPE_k)^{DP}$ for each $1 \le j \le n$ and each $1 \le k \le n$
DisjointDataProperties(DPE_1 ... DPE_n)	$(DPE_j)^{DP} \cap (DPE_k)^{DP} = \varnothing$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
DataPropertyDomain(DPE CE)	$\forall\, x, y$: $(x, y) \in (DPE)^{DP}$ implies $x \in (CE)^{C}$
DataPropertyRange(DPE DR)	$\forall\, x, y$: $(x,$ y$) \in (DPE)^{DP}$ implies $y \in (DR)^{DT}$
FunctionalDataProperty(DPE)	$\forall\, x, y_1, y_2$: $(x, y_1) \in (DPE)^{DP}$ and $(x, y_2) \in (DPE)^{DP}$ imply $y_1 = y_2$

A.2.4 Datatype Definitions

Satisfaction of datatype definitions in Γ with respect to an ontology O is defined as shown in Table 7.

Table 7. Satisfaction of Datatype Definitions in an Interpretation

in an Interpretation

Axiom	Condition
DatatypeDefinition(DT DR)	$(DT)^{DT} = (DR)^{DT}$

A.2.5 Keys

Satisfaction of keys in Γ with respect to an ontology O is defined as shown in Table 8.

Table 8. Satisfaction of Keys in an Interpretation

Axiom	Condition
HasKey(CE (OPE$_1$ ... OPE$_m$) (DPE$_1$ ... DPE$_n$))	$\forall\ x, y, z_1, ..., z_m, w_1, ..., w_n$: if $x \in (CE)^C$ and $ISNAMED_O(x)$ and $y \in (CE)^C$ and $ISNAMED_O(y)$ and $(x, z_i) \in (OPE_i)^{OP}$ and $(y, z_i) \in (OPE_i)^{OP}$ and $ISNAMED_O(z_i)$ for each $1 \le i \le m$ and $(x, w_j) \in (DPE_j)^{DP}$ and $(y, w_j) \in (DPE_j)^{DP}$ for each $1 \le j \le n$ then x = y

A.2.6 Assertions

Satisfaction of OWL 2 assertions in Γ with respect to an ontology O is defined as shown in Table 9.

Table 9. Satisfaction of Assertions in an Interpretation

Axiom	Condition
SameIndividual(a$_1$ ... a$_n$)	$(a_j)^I = (a_k)^I$ for each $1 \le j \le n$ and each $1 \le k \le n$
DifferentIndividuals(a$_1$ ... a$_n$)	$(a_j)^I \ne (a_k)^I$ for each $1 \le j \le n$ and each $1 \le k \le n$ such that $j \ne k$
ClassAssertion(CE a)	$(a)^I \in (CE)^C$
ObjectPropertyAssertion(OPE a$_1$ a$_2$)	$((a_1)^I, (a_2)^I) \in (OPE)^{OP}$
NegativeObjectPropertyAssertion(OPE a$_1$ a$_2$)	$((a_1)^I, (a_2)^I) \notin (OPE)^{OP}$
DataPropertyAssertion(DPE a lt)	$((a)^I, (lt)^{LT}) \in (DPE)^{DP}$
NegativeDataPropertyAssertion(DPE a lt)	$((a)^I, (lt)^{LT}) \notin (DPE)^{DP}$