Knowledge Representation and Reasoning on the Semantic Web: OWL

Ian Horrocks[1] and Peter F. Patel-Schneider[2]

[1]Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK
Ian.Horrocks@comlab.ox.ac.uk

[2]Alcatel-Lucent Bell Labs Research
Murray Hill, New Jersey, USA
pfps@research.bell-labs.com

Abstract. OWL is the ontology language recommended by the W3C. OWL is heavily based on the knowledge representation languages called Description Logic, which provide the basic representation features of OWL. OWL also includes facilities that integrate it into the mainstream of the Web, including use of IRIs as names, XML Schema datatypes, and ontologies as Web documents, which can then import other OWL ontologies over the Web. Because OWL is based on Description Logics, its constructs have a well defined meaning and there are tools that effectively perform inference within OWL, enabling the discovery of information that is not explicitly stated in OWL ontologies.

1 Introduction

A major feature of the Semantic Web is the ability to provide definitions for objects and types of objects that are accessable and manipulable from within the Semantic Web. In computer science, a collection of these sorts of definitions about a particular domain is called an ontology, although philosophers may (and probably will) have a different understanding of what constitutes an ontology.

It is this ability to provide machine-processable definitions and meanings that most divides the Semantic Web from the Visual Web of HTML, CSS, SVG, etc., which is focussed on presenting information to humans, and the Syntactic Web of XML, which is focussed on the transfer of uninterpreted data between applications (which themselves are written by humans who have extra-Web knowledge of what the data means).

OWL [62, 30] is an ontology language designed for use in the Semantic Web and is the language recommended by the W3C for this use. OWL has influences from quite a number of sources, but its main representational facilities are directly based on Description Logics [1]. This basis confers upon OWL a logical framework, including both syntax and model-theoretic semantics. OWL also inherits from Description Logics a concern for practical reasoning and effective,

readily available reasoners, for example the Description Logic reasoners Pellet [74] and HermiT [51], both of which have been extended to handle all of OWL.

OWL is also completely integrated into the Semantic Web and uses other W3C recommendations. The official transfer syntax for OWL ontologies is RDF/XML, a syntax for transferring RDF graphs. Names (of individuals, etc.) in the Semantic Web are IRIs [66], so names in OWL are IRIs. OWL uses XML Schema datatypes [9] to define and type the data values that it uses. OWL ontologies are Web documents and are stored and retrieved just as other Web documents. OWL includes constructs for identifying ontologies and importing one ontology into another. OWL ontologies can also be combined with rules using the new W3C Rule Interchange Format (RIF) standard [67].

The remainder of this chapter will: discuss the history of OWL, and the various influences that have shaped it; describe the results of these influences, namely the OWL language; and explain how OWL is used in applications, and in particular how OWL reasoning services are used in practice. The focus will be on OWL 2 [56], an extension and revision of OWL that became a W3C Recommendation on the 27th of October, 2009.

2 History and Influences

2.1 Description Logics

Because OWL is heavily based on Description Logics, it inherits their history and shares their influences.

The history of Description Logics started with attempts to formalize Semantic Nets, which themselves were attempts to provide a sort of natural representation based on labelled graphs (nets). A major problem with Semantic Nets was that there was no formal meaning for their graphs, leading to conflicts over just what complex nets meant when divorced from a particular system that provided some data manipulation facilities for Semantic Nets. Another early influence on Description Logics were frame systems, such as FRL [68], which had many of the characteristics of Semantic Nets, but grouped related information together into a frame.

The representation language KL-ONE [10, 11] was a knowledge representation system that had many of the characteristics of Semantic Nets and frame systems. Not long after its inception there were attempts to provide a full formal semantics for KL-ONE [40]. KL-ONE with this semantics can be considered as the first proto-Description Logic.

It was then quickly determined that the systems that manipulated KL-ONE constructs were incomplete with respect to the semantics i.e., they systematically were unable to make conclusions that were implicit in the information given to them; shortly thereafter it was determined that *complete* inference for KL-ONE was very difficult, in fact undecidable [72], and that inference even in very limited subsets of KL-ONE was worst-case intractable [40]. This lead to the development of simpler representation systems, like KANDOR [61]. The aim for KANDOR was

to design a useful representation language in which complete inference would be easy (both easy to implement and easy to decide). Unfortunately, the designers of KANDOR were able to achieve easy complete inference only by imposing severe limits on its expressive power.

The Description Logic community then split into three groups. One group continued work on simple Description Logics with complete or nearly complete but worst-case-tractable reasoning algorithms, such as the one implemented in the CLASSIC system [63]. Another group continued work on more expressive Description Logics that had worst case intractable reasoning, such as those implemented in the KRIS and CRACK systems [3, 12]. This group was more interested in the formal aspects of these Description Logics. A third group built partial reasoners for very expressive, i.e., undecidable, Description Logics, such as the one implemented in the LOOM system [45].

Eventually, with the FACT system [27], it was shown that it was possible to provide a reasoner for an expressive Description Logic ($\mathcal{SH}$) that, in spite of the worst case intractability of basic reasoning problems, nonetheless provided complete and effective reasoning in most cases. This invigorated the community, as now Description Logics that contained useful representational constructs could be provided with effective reasoners and thus could be used in applications.

In spite of this success, continuing concerns about tractability, particularly in applications requiring very large ontologies or data sets, have rekindled interest in smaller Description Logics where reasoning is worst case tractable. Research in this area has focused on two main families of Description Logics: the $\mathcal{EL}$ family and the DL-Lite family. The advantage of these logics is that standard reasoning problems have either PTime complexity (for members of the $\mathcal{EL}$ family) or are in AC^0 (for members of the DL-Lite family).

***SROIQ*(D)** Modern Description Logics are generally quite alike, with most using the same syntax for their constructs and the same First Order semantics. Even the names of many of the modern Description Logics are rather similar as they are built up from letters that signify features of the logic.

The Description Logics that underlie the various Semantic Web ontology languages are all extensions to the well known Description Logic $\mathcal{ALC}$ [73] plus transitively closed primitive roles [70]. This Description Logic was called $\mathcal{S}$ due to its relationship with the propositional (multi) modal logic $\mathbf{S4_m}$ [71] (it can also be called $\mathcal{ALC}_{R^+}$, but this name is cumbersome to add letters to representing additional features). $\mathcal{S}$ is then extended to include features such as role inclusion axioms ($\mathcal{H}$, or $\mathcal{R}$ if role chains are allowed), nominals ($\mathcal{O}$), inverse roles ($\mathcal{I}$), and number restrictions ($\mathcal{Q}$ if qualified, $\mathcal{N}$ if not).

The syntax and semantics of these features is summarised in Figure 1, where A is a concept name, C and D are concepts, R and S are roles, $\mathbf{R}_+$ is the set of transitive roles, o is an individual name, P is a simple role (i.e., one that is not transitive and has no transitive sub-role), and n is a non-negative integer. Note that, in order to retain decidability, it is necessary to impose some restrictions on the property hierarchy and on the use of roles in number restrictions; for example,

transitive roles cannot be used in number restrictions. A complete description of these languages and the relevant restrictions on roles can be found in the Description Logic literature [33, 32, 29].

Construct Name	Syntax	Semantics	
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	$\mathcal{S}$
atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	
transitive role	$R \in \mathbf{R}_+$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^+$	
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	
exists restriction	$\exists R.C$	$\{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\}$	
value restriction	$\forall R.C$	$\{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \text{ implies } y \in C^{\mathcal{I}}\}$	
role hierarchy	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{H}$
role chains	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$	$\mathcal{R}$
nominal	$\{o\}$	$\{o^{\mathcal{I}}\}$	$\mathcal{O}$
inverse role	R^-	$\{\langle x, y\rangle \mid \langle y, x\rangle \in R^{\mathcal{I}}\}$	$\mathcal{I}$
number restriction	$\geqslant n\, P$	$\{x \mid \sharp\{y.\langle x, y\rangle \in P^{\mathcal{I}}\} \geqslant n\}$	$\mathcal{N}$
	$\leqslant n\, P$	$\{x \mid \sharp\{y.\langle x, y\rangle \in P^{\mathcal{I}}\} \leqslant n\}$	
qualified number restriction	$\geqslant n\, P.C$	$\{x \mid \sharp\{y.\langle x, y\rangle \in P^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geqslant n\}$	$\mathcal{Q}$
	$\leqslant n\, P.C$	$\{x \mid \sharp\{y.\langle x, y\rangle \in P^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leqslant n\}$	

Fig. 1. Syntax and semantics of the $\mathcal{S}$ family of Description Logics

These logics can also be extended with *concrete domains*, which allow for the use of "concrete" types, such as the integers, values, such as the integer "5", and predicates such as integer comparisons [2, 41, 43]. A simplified form of concrete domains, known as Datatypes, is denoted by appending ($\mathbf{D}$) to the name of the logic, e.g., $\mathcal{SHOIN}(\mathbf{D})$ [31]; Datatypes restrict the interactions between concrete and "abstract" parts of a knowledge base so as to avoid problems of undecidability and to simplify implementation, and are widely used in ontology languages, including OWL and OWL 2 [49]. The syntax and semantics of Datatypes is summarised in Figure 2, where D is a datatype name, T is a concrete role, v is a data value and n is a non-negative integer.

2.2 OIL and DAML+OIL

The first significant effort to build a language combining Description Logics and the Semantic Web was OIL (the Ontology Inference Layer) [28], which was developed within the On-To-Knowledge research project (see `http://www.ontoknowledge.org/`) funded by the European Union. The OIL language was explicitly designed as "a web-based representation and inference language for ontologies [that combines] the widely used modeling primitives from frame-based

Construct Name	Syntax	Semantics
datatype	D	$D^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}_{\mathbf{D}}$
data value	v	$v^{\mathcal{I}} \in \Delta^{\mathcal{I}}_{\mathbf{D}}$
concrete role	T	$^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}_{\mathbf{D}}$
enumerated datatype	$\{v_1, \ldots, v_n\}$	$\{v_1^{\mathcal{I}}, \ldots, v_n^{\mathcal{I}}\}$
exists restriction	$\exists T.D$	$\{x \mid \exists y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\}$
value restriction	$\forall T.D$	$\{x \mid \forall y.\langle x, y\rangle \in T^{\mathcal{I}} \text{ implies } y \in D^{\mathcal{I}}\}$
number restriction	$\geqslant n\, T$	$\{x \mid \sharp\{y.\langle x, y\rangle \in T^{\mathcal{I}}\} \geqslant n\}$
	$\leqslant n\, T$	$\{x \mid \sharp\{y.\langle x, y\rangle \in T^{\mathcal{I}}\} \leqslant n\}$
qualified number restriction	$\geqslant n\, T.D$	$\{x \mid \sharp\{y.\langle x\rangle y \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\} \geqslant n\}$
	$\leqslant n\, T.D$	$\{x \mid \sharp\{y.\langle x\rangle y \in T^{\mathcal{I}} \text{ and } y \in D^{\mathcal{I}}\} \leqslant n\}$

Fig. 2. Syntax and semantics of Description Logic Datatypes

languages with the formal semantics and reasoning services provided by description logics" (see `http://www.ontoknowledge.org/oil/`). OIL was so closely tied to Description Logics that the semantics was specified as a mapping from its syntax to the Description Logic $\mathcal{SHIQ}$ [33, 17]. From this mapping, OIL gained both a clear semantics and a means to exploit the reasoning services of Description Logic systems such as FaCT [27], RACER [22] and Pellet [64] that implement (most of) $\mathcal{SHIQ}(\mathcal{D})$. OIL was also influenced by frame systems as its syntax provided for the grouping of constructs related to a particular individual or class, thus providing the appearance of a single construct per individual or class.

There were three syntaxes for OIL: a text based syntax, an XML syntax and a syntax based on RDF (Resource Description Framework—see `http://www.w3.org/RDF/`). The first of these was meant for presentation use only, allowing OIL ontologies to be easily viewed without the verbose additions needed in the other two syntaxes. The XML and RDF syntaxes were suitable for transferring OIL ontologies in the Semantic Web, the latter allowing OIL ontologies to be written directly as RDF graphs. OIL even used the same set of names as RDF (i.e., `rdfs:Class`, `rdfs:subClassOf`, etc.), permitting non-OIL RDF to be used along with OIL ontologies. So in OIL there was, for the first time, the adaptation of Description Logic constructs for the Semantic Web. OIL, however, did not have complete integration with RDF. Some of the Description Logic constructs in OIL, particularly exists, value, and number restrictions, were not RDF classes but instead had a frame-like syntax, and were not RDF classes.

The DARPA Agent Markup Language (DAML) program then joined into the above effort to update and modify OIL to provide an even closer relationship to the Semantic Web, particularly RDFS. The result of this collaboration was DAML+OIL (see `http://www.w3.org/Submission/2001/12/`).

From the point of view of language constructs, the differences between OIL and DAML+OIL are relatively trivial. Although there is some difference in "keyword" vocabulary, there is usually a one-to-one mapping of constructors, and in the cases where the constructors are not completely equivalent, simple

translations are possible. However, the frame structure of OIL's syntax is much less evident in DAML+OIL, with the result that a DAML+OIL ontology is more Description-Logic-like in that it consists largely of a relatively unstructured collection of subsumption and equality axioms. This can make it more difficult to use DAML+OIL with frame-based tools such as Protégé [21] or OilEd [7], because the axioms may be susceptible to many different frame-like groupings [6].

The initial release of DAML+OIL did not include any specification of datatypes. The language was, however, subsequently extended with arbitrary datatypes from the XML Schema type system [9], which can be used in restrictions (slot constraints) and range constraints. As in $\mathcal{SHOQ}(\mathcal{D})$ [31], a clean separation is maintained between instances of "object" classes (defined using the ontology language) and instances of datatypes (defined using the XML Schema type system). In particular, it is assumed that the domain of interpretation of object classes is disjoint from the domain of interpretation of datatypes, so that an instance of an object class (e.g., the individual Italy) can never have the same interpretation as a value of a datatype (e.g., the integer 5), and that the set of object properties (which relate individuals to individuals) is disjoint from the set of datatype properties (which relate individuals to datatype values).

2.3 OWL and OWL 2

DAML+OIL did solve the problem of the syntactic disconnect with RDFS; there was, however, no semantic integration between DAML+OIL and RDF—in fact RDF did not yet have a formal semantics. Further, DAML+OIL was not recommended by any organization beyond the two projects that had sponsored its development.

Therefore some of the people involved in the design of DAML+OIL decided to try to create a W3C recommendation for an ontology language to be tightly integrated into the Semantic Web. To this end DAML+OIL was submitted to the W3C (see `http://www.w3.org/Submission/2001/12/`), and a Web Ontology Working Group (WebOnt) was subsequently chartered to produce a Semantic Web Ontology Language based on DAML+OIL, but more tightly integrated with RDF and RDFS as well as other web standards. At the same time the W3C RDF Core Working Group was, among other things, developing a formal semantics for RDF and RDFS [24].

The result of WebOnt was OWL, the W3C Web Ontology Language (see `http://www.w3.org/2004/OWL/`). OWL had an official Semantic Web syntax in the form of RDF graphs, and could thus be transferred and manipulated using Semantic Web tools There was also an internal syntax for OWL, which was easier to manipulate formally, and there soon came to be a compact frame-like syntax for OWL called the Manchester Syntax [26]. OWL had a model-theoretic semantics compatible with the semantics of Description Logics, so Description Logics reasoners could, in principle, provide reasoning facilities for OWL as they did for OIL and DAML+OIL. However, the Description Logic $\mathcal{SHOIN}(\mathbf{D})$ on which OWL was based was more expressive than those supported by existing

reasoners, and it wasn't until more than a year after the completion of OWL that suitable reasoning algorithms and implementations became available.

OWL used the vocabulary of RDF and RDFS where possible, so RDF and RDFS tools could process OWL ontologies that fit into their limited expressive power, as they could for DAML+OIL. OWL's model-theoretic semantics was fine-tuned to be as compatible as possible with the new semantics for RDF and RDFS so that the constructs common with RDF and RDFS had compatible meaning. OWL included mechanisms to import other ontologies and Semantic Web documents across the Semantic Web; it also included a way of adding extra-logical information into ontologies to be used, for example, to record the status of particular classes or individuals.

To obtain the desirable implementation characteristics of Description Logics in OWL it was necessary to limit the ways in which some constructs of RDF and RDFS were used. This required, in particular, restricting the use of RDF and RDFS syntax to that which corresponded to syntactically coherent OWL, ruling out both malformed OWL syntax and the use of RDF, RDFS and OWL vocabulary as individual, class or property names in an OWL ontology. It additionally required vocabulary separation, i.e., ensuring that the sets of names used for classes, properties and individuals were disjoint. Ontologies satisfying these restrictions were guaranteed to have the desirable characteristics of Description Logics, and were called OWL DL ontologies. Ontologies that violated these restrictions were called OWL Full ontologies. Description Logic reasoners would not be complete for OWL Full ontologies, and, indeed, reasoning in OWL Full is trivially undecidable (see [30] for a detailed discussion). The advantage of OWL Full was that any RDF or RDFS document could be used as an OWL Full ontology.

Although very successful, OWL did not, of course, satisfy all user requirements. After extensive discussions between users, theoreticians and implementers, in particular at the 2005 OWL Experiences and Directions Workshop (see `http://www.mindswap.org/2005/OWLWorkshop/`), it was decided to address some of these requirements via an incremental revision of OWL, provisionally called OWL 1.1. The initial goal of OWL 1.1 was to exploit recent developments in Description Logic research in order to address some of the expressivity limitations of the OWL. However, as the design of OWL 1.1 progressed, it was decided to also address performance requirements by exploiting research into smaller Description Logics with desirable computational properties.

The development of OWL 1.1 was initially undertaken by an informal group of language users and developers. After the original specification reached a mature state, and first implementations were released, the OWL 1.1 proposal was submitted to the W3C as a Member Submission (see `http://www.w3.org/Submission/2006/10/`); this was then taken as a starting point for a new W3C Working Group that was officially formed in September 2007. As work on the new language progressed, the initial Member Submission evolved significantly, and the Working Group eventually decided to call the new language OWL 2 so as to indicate a substantial step in the evolution of the language.

OWL 2 is based on $\mathcal{SROIQ}(\mathbf{D})$ and so extends OWL with qualified cardinality restrictions and with significantly extended expressivity w.r.t. properties, e.g., the ability to assert that properties are reflexive, irreflexive, asymmetric and disjoint, and the ability to compose properties into property chains. OWL 2 also weakens the name separation restriction imposed in OWL—in OWL 2 the same name can be used for a class, a property and an individual, a feature known as punning. In addition, OWL 2 provides greatly extended support for datatypes, including many of the XML Schema datatypes and facets, and for annotations, including, e.g., the ability to annotate axioms as well as entities. Finally, OWL 2 also provides a limited form of database style keys.

As well as increasing the expressive power of the complete language, OWL 2 also defines several *profiles*: language fragments that have desirable computational properties (see `http://www.w3.org/TR/owl2-profiles/`). These include a profile based on DL Lite, a Description Logic for which standard reasoning problems can be reduced to SQL query answering; a profile based on $\mathcal{EL}^{++}$, a Description Logic for which standard reasoning problems can be performed in polynomial time; and a profile based on DLP [20], a logic for which query answering can be implemented using rule based techniques that have been shown to scale well in practice. The old OWL Lite profile was depricated as it provides no significant computational advantage w.r.t. OWL.

In addition to language features, OWL 2 boasts several "convenience" features, including an improved specification in both BNF and UML formats, a fully validated XML Syntax and the use of Manchester syntax as a text based and more human friendly syntax.

3 The OWL 2 Language

Because OWL is heavily based on Description Logics, OWL 2 exhibits many characteristics of typical Description Logics. In particular, it describes the domain in terms of individuals, classes (called *concepts* in Description Logics), properties (called *roles* in Description Logics), and data types and values (called *concrete domains* in Description Logics). Individual names, e.g., "John", refer to elements of the domain; concepts, e.g., "University", describe sets of individuals having similar characteristics; roles, e.g., "studiesAt", describe relationships between pairs of individuals (such as "John studiesAt Oxford"); and data types, e.g., "integer", describe sets of data values, e.g., "18". Class descriptions can also be composed from all of the above components using various constructors, including, for example, the Booleans.

Like a Description Logic knowledge base, an OWL 2 ontology consists of a set of axioms and facts that describe the domain, for example, asserting that GradStudent is a subCalssOf Student, that John is a Student or that John hasAge 18. Finally, like a Description Logic, OWL 2 can be seen as a fragment of First Order Logic (FOL), and is given a formal semantics based on First Order model theory (although it could equally well be given via a translation into Description Logic, or even directly into FOL).

Because OWL 2 is an ontology language for the Semantic Web it has some differences from most Description Logics and does some things in different ways from Description Logics.

These differences start with the names used in OWL 2, which are IRIs, the names that underlie the Semantic Web (and indeed the Web itself) [66]. As IRIs tend to be very long, OWL 2 syntaxes provide facilities for short forms of names, roughly the same as QNames used by SPARQL [77]. So, for example, OWL 2 syntaxes allow owl:Thing as a short form of `http://www.w3.org/2002/07/owl#Thing`. OWL 2 also allows anonymous individuals (individuals without global names), written out as in the RDF syntax for blank nodes, e.g., `_:id`. OWL 2 does not assume that different names refer to different entities, so, for example, `ex:Jack` and `ex:John` can both be names for the same person (this is discussed in more detail in Section 3.5); nor does it assume that the names used for individuals, classes and properties are disjoint (as was the case in OWL), so, for example, the same name could be used to denote both a class and an individual (this is discussed in more detail in Section 3.3).

Moreover, OWL 2 largely uses the datatype facilities of XML Schema [9], including floating point numbers, instead of the more mathematical (and easier to work with) datatypes common to most Description Logics. The set of supported datatypes and facets is defined in the OWL 2 datatype map, which will be discussed in more detail in Section 3.2.

OWL 2 has several syntaxes. The standard syntax of the Semantic Web, RDF/XML [8] , is the one syntax that all OWL 2 implementations must support. However, as RDF/XML is very verbose and very hard to read, there are other syntaxes for OWL 2, including an XML syntax for integration with XML tools, a functional-style syntax [59] that is used for precision and in formal documents, and an easier to read syntax designed for presentation, called the Manchester syntax [55]. Manchester syntax will be used in the remainder of this chapter, precisely because it is designed for presentation to humans.

3.1 OWL 2 Ontologies

OWL 2 has the notion of an ontology—meant to be a collection of related information that describes a domain. These ontologies can (and generally are) stored as Web documents and can be combined into larger collections of information. In contrast to a Description Logic knowledge base, where conceptual and instance level statements are usually separated into, respectively, a set of Tbox axioms and a set of Abox assertions, an OWL 2 ontology consists of a single set of axioms that include both conceptual and instance level statements. OWL is non-standard in this regard: ontologies are more typically thought of as describing only the structure of a domain (in terms of classes and properties), and not as describing a particular situation (in terms of instances of classes and properties); in this more common usage, an ontology is therefore equivalent to a Description Logic Tbox, and not to the combination of a Tbox and an Abox.

In addition to the set of axioms, an ontology may be named using an IRI, and different versions of the same ontology may additionally be named with a

version IRI. An ontology may also import other OWL 2 ontologies identified by their ontology or version IRIs. The set of axioms that constitute an ontology is taken to be equal to the union of the set of axioms contained in the ontology and the sets of axioms that constitute each of the ontologies that it imports; this is sometimes referred to explicitly as the *imports closure* of an ontology. Note that this definition is recursive: the imported ontologies could themselves import other ontologies, and so on.

Finally, an ontology can also be annotated with information such as the creator's name and copyright information. Annotation properties are used for this purpose, and there are several built-in annotation properties intended for use with ontologies; these include `owl:priorVersion`, `owl:backwardCompatibleWith` and `owl:incompatibleWith`, all used to specify prior versions of the ontology, and optionally to describe their compatibility with the current ontology. More will be said about annotations and annotation properties in Section 3.6.

3.2 OWL 2 Datatypes

OWL 2 uses datatypes from XML Schema datatypes, so `xsd:integer` is a datatype in OWL, namely the type of integers; datatype restrictions using facets are also allowed, as in `xsd:integer xsd:minInclusive "5"^^xsd:integer` (the integers greater than or equal to 5). The set of supported datatypes and facets is defined in the OWL 2 datatype map [59].

For data values (integers, strings, etc.) OWL 2 uses the syntax of RDF, so `"2"^^xsd:integer` is the way to write the integer 2, and `"23.5"^^xsd:decimal` is the way to write a decimal number. To enhance human readability for typical data values, the OWL 2 Manchester syntax allows strings, integers, decimals, and floats to be written as in most programming languages, as in `"abc"`, `25`, `25.55`, and `25.55F`. OWL 2 also allows plain RDF literals, which are a combination of a string and an optional language tag and can be written as in RDF, e.g., `"favor"@en-us`. These plain literals belong to the datatype `rdf:PlainLiteral`.

3.3 OWL 2 Entities

As mentioned above, OWL 2 uses IRIs as names for classes, properties, individuals and datatypes; collectively, these names are known as *entities*. In a Description Logic, the set of entities occurring in an ontology is usually called the signature. The entities, together with data values, make up the basic building blocks of OWL 2 ontologies.

In OWL 2 the properties are divided into three disjoint sets of object properties, data properties and annotation properties. Object properties are used to relate one individual to another; for example, the object property `ex:worksFor` might be used to relate a person to a company. Data properties are used to relate an individual to a data value; for example, the data property `ex:hasAge` might be used to relate a person to an integer value. Finally, annotation properties are

used to add uninterpreted information (such as textual comments) to individuals, classes, properties and ontologies; more will be said about annotations in Section 3.6.

Declarations and typing The entities used in an OWL 2 ontology can, and in some cases must, be typed using a suitable declaration. Declarations are used to avoid possible ambiguities and to ensure the required separation of object, data and annotation property names. For example, declarations could be used as follows to state that `ex:worksFor`, `ex:hasAge` and `ex:authoredBy` are, respectively, object, data and annotation properties:

```
ObjectProperty: ex:worksFor
DataProperty: ex:hasAge
AnnotationProperty: ex:authoredBy
```

Punning In OWL a given name could be used to refer to only a single type of entity. In contrast, OWL 2 allows the same name to be used to refer to different types of entity; for example, `ex:Eagle` might be used to denote both the class (a subclass of `ex:Bird`) and an individual (an instance of `ex:Species`):

```
Class: ex:Eagle
  SubClassOf: ex:Bird
Individual: ex:Eagle
  Types: ex:Species
```

On the face of it this would seem to take OWL 2 beyond the strictly first order realm of Description Logics. However, the semantics of OWL 2 is designed so that the interpretations of a name used as different entity types are totally unconnected; the interpretation of `ex:Eagle` the individual is, for example, totally unconnected to the interpretation of `ex:Eagle` the class. In effect, entity names are treated as though their type is part of the name, so the above example could be read as:

```
Class: ex:Eagle_C
  SubClassOf: ex:Bird
Individual: ex:Eagle_I
  Types: ex:Species
```

This re-use of names, but with different meanings, is called *punning*. OWL 2 does place some restrictions on the use of punning: as discussed in Section 3.3, punning between object, data and annotation properties is not allowed; in addition, punning between classes and datatypes is disallowed.

3.4 Expressions

As in a Description Logic, OWL 2 class and property expressions generalise classes and properties, while data ranges generalise datatypes. In particular, a

Manchester Syntax	DL Syntax
P	P
`inverse` R	R^-

Fig. 3. OWL 2 Object Property Expressions

property is a property expression, and property expressions can be combined using various operators to form new property expressions; a datatype is a data range, and data ranges can be combined using various operators (including facet based restrictions) to form new data ranges; and a class is a class expression, and class expressions, property expressions and data ranges can be combined using various operators to form new class expressions.

Property Expressions An object property expression in OWL 2 can be either an object property name or an expression constructed using the available operators as shown in Figure 3, where P is an object property name (an IRI), and R is an arbitrary property expression. As can be seen, there is only one constructor for use with object property expressions: inverse. Arbitrary nesting of inverses is in principle possible, but as `inverse (inverse` R`)` is equivalent to R, it is reasonable to assume that all OWL 2 object property expressions are of the form P or `inverse` P for P an object property name (an IRI).

The set of constructors available for forming datatype properties is even more limited: there are none! All datatype property expressions are, therefore, of the form U for U a datatype property name (an IRI).

Manchester Syntax	DL Syntax
B	B
D_1 `and ...and` D_n	$D_1 \cap \ldots \cap D_n$
D_1 `or ...or` D_n	$D_1 \cup \ldots \cup D_n$
not D	$\neg D$
$\{v_1 \ldots v_n\}$	$\{v_1\} \cup \ldots \cup \{v_n\}$

Fig. 4. OWL 2 Data Ranges

Data Ranges In OWL 2 the supported XML schema datatypes (including datatype restrictions using facets) are data ranges, and data ranges can also be formed from other data ranges and values using various constructors as shown in Table 4, where B is an XML schema datatype or datatype restriction, D (possibly subscripted) is an arbitrary data range, and v_i is a data value. For example, a data range could be defined by using an XML schema facet to restrict a base type, as in `xsd:integer xsd:maxExclusive "10"^^xsd:integer` (the

Manchester Syntax	DL Syntax
A	A
`owl:Thing`	$\top$
`owl:Nothing`	$\bot$
C_1 `and` ...`and` C_n	$C_1 \sqcap \ldots \sqcap C_n$
C_1 `or` ...`or` C_n	$C_1 \sqcup \ldots \sqcup C_n$
`not` C	$\neg C$
$\{o_1 \ldots o_n\}$	$\{o_1\} \sqcup \ldots \sqcup \{o_n\}$
R `some` C	$\exists R.C$
R `only` C	$\forall R.C$
R `value` o	$\exists R.\{o\}$
R `self`	$\exists R.self$
R `min` n	$\geqslant n\, R$
R `max` n	$\leqslant n\, R$
R `exactly` n	$\geqslant n\, R \sqcap \leqslant n\, R$
R `min` n C	$\geqslant n\, R.C$
R `max` n C	$\leqslant n\, R.C$
R `exactly` n C	$\geqslant n\, R.C \sqcap \leqslant n\, R.C$
U `some` D	$\exists U.D$
U `only` D	$\forall U.D$
U `value` v	$\exists U.\{v\}$
U `min` n	$\geqslant n\, U$
U `max` n	$\leqslant n\, U$
U `exactly` n	$\geqslant n\, U \sqcap \leqslant n\, U$
U `min` n D	$\geqslant n\, U.D$
U `max` n D	$\leqslant n\, U.D$
U `exactly` n D	$\geqslant n\, U.D \sqcap \leqslant n\, U.D$

Fig. 5. OWL 2 Class Descriptions

integers less than 10); it could be defined by enumerating a set of values, as in `{"1"^^xsd:integer "2"^^xsd:integer "3"^^xsd:integer}` (the integers 1, 2 and 3); or it could be defined by using one of the Boolean operators to combine other data ranges, as in `xsd:integer or xsd:float` (the union of the integers and floats).

Class Expressions As mentioned above, OWL 2 is very closely related to $\mathcal{SROIQ}(\mathbf{D})$, and provides a correspondingly wide range of operators for building class expressions. These are summarised in Figure 5, where A is a class name (an IRI), C (possibly subscripted) is an arbitrary class expression, o_i is an individual name (an IRI), R is an object property expression, U is a datatype property expression, D is a data range, v is a data value, and n is a nonnegative integer. These can be used to form descriptions characterizing sets of individuals. For example, `ex:Person and ex:hasChild some ex:Person` describes those individuals that are instances of `ex:Person` and are related via the property `ex:hasChild` to an instance of `ex:Person` (i.e., parents).

Like $\mathcal{SROIQ}(\mathbf{D})$, OWL 2 supports the standard Boolean constructors (`and`, `or` and `not`), which correspond directly to $\sqcap$, $\sqcup$ and $\neg$ in Description Logic. The OWL 2 OneOf constructor allows a class to be formed by enumerating its instances, written $\{o_1 \ldots o_n\}$ in the Manchester Syntax, and equivalent to a disjunction of nominals $\{o_1\} \sqcup \ldots \sqcup \{o_n\}$ in $\mathcal{SROIQ}(\mathbf{D})$. The built-in `owl:Thing` and `owl:Nothing` classes correspond directly to $\top$ and $\bot$ in Description Logic.

OWL 2 also supports the full set of $\mathcal{SROIQ}(\mathbf{D})$ restrictions, including exists and value restrictions (`some` and `only` in the Manchester Syntax) and both qualified and unqualified maximum, minimum and exact cardinality restrictions. Note that exact cardinality restrictions are equivalent to a symmetrical pair of Description Logic minimum and maximum cardinality restrictions, and that the HasValue restriction in OWL 2 (written R `value` o in Manchester Syntax) is simply shorthand for an exists restriction with a nominal as the restricting class. A similar set of restrictions can be used with datatypes and data values.

3.5 Axioms

OWL 2 axioms provide information about classes, properties, data ranges, keys and individuals, as shown in Figure 6, where A is a class name (an IRI), C (possibly subscripted) is an arbitrary class, P is an object property name (an IRI), R (possibly subscripted) is an arbitrary object property, T is a data property name (an IRI), S (possibly subscripted) is an arbitrary data property, D is a data range, B is a datatype (an IRI), U (possibly subscripted) is a property (either object or data), and i (possibly subscripted) is an individual. The axioms as presented here mirror the structural specification and the RDF/XML and XML syntaxes; using the Manchester Syntax, however, it is also possible to group together statements about classes, properties and individuals. For example, the following "class frame" could be used to define cricket fans as people who like cricket while at the same time asserting that cricket fans drink nothing but beer and that no individual can be both a cricket fan and a baseball fan:

```
Class: CricketFan
  EquivalentTo: ex:Person that ex:likes some ex:Cricket
  SubClassOf: ex:drinks only ex:Beer
  DisjointWith: ex:BaseballFan
```

This would be equivalent to the DL axioms:

$$\texttt{CricketFan} \equiv \texttt{ex:Person} \sqcap \exists \texttt{ex:likes.ex:Cricket}$$
$$\texttt{CricketFan} \sqsubseteq \forall \texttt{ex:drinks.ex:Beer}$$
$$\texttt{CricketFan} \sqsubseteq \neg \texttt{ex:BaseballFan}$$

Similarly, statements about a given individual could be combined in Manchester Syntax as follows:

```
Individual: Canada
  Types: ex:Country
```

Manchester Syntax	DL Syntax
`Class:` A `SubClassOf:` C	$A \sqsubseteq C$
`Class:` A `EquivalentTo:` C	$A \equiv C$
`EquivalentClasses:` $C_1, \ldots, C_n$	$C_i \equiv C_{i+1}$ `for` $1 \leq i < n$
`DisjointClasses:` $C_1, \ldots, C_n$	$C_i \sqsubseteq \neg C_j$ `for` $1 \leq i < j \leq n$
`Class:` A `DisjointUnionOf:` $C_1, \ldots, C_n$	$C_i \sqsubseteq \neg C_j$ `for` $1 \leq i < j \leq n$
	$A \equiv C_1 \sqcup \ldots \sqcup C_n$
`ObjectProperty:` P `SubPropertyOf:` R	$P \sqsubseteq R$
`ObjectProperty:` P `EquivalentTo:` R	$A \equiv C$
`EquivalentProperties:` $R_1, \ldots, R_n$	$R_i \equiv R_{i+1}$ `for` $1 \leq i < n$
`DisjointProperties:` $R_1, \ldots, R_n$	$R_i \sqsubseteq \neg R_j$ `for` $1 \leq i < j \leq n$
`ObjectProperty:` P `InverseOf:` R	$P \equiv R^-$
`ObjectProperty:` P `Domain:` C	$\exists P.\top \sqsubseteq C$
`ObjectProperty:` P `Range:` C	$\top \sqsubseteq \forall P.C$
`ObjectProperty:` P `Characteristics: Functional`	$\top \sqsubseteq \leqslant 1\, P$
`ObjectProperty:` P `Characteristics: InverseFunctional`	$\top \sqsubseteq \leqslant 1\, P^-$
`ObjectProperty:` P `Characteristics: Reflexive`	$\top \sqsubseteq \exists P.self$
`ObjectProperty:` P `Characteristics: Irreflexive`	$\exists P.self \sqsubseteq \bot$
`ObjectProperty:` P `Characteristics: Symmetric`	$P \equiv P^-$
`ObjectProperty:` P `Characteristics: Asymmetric`	
`ObjectProperty:` P `Characteristics: Transitive`	$P \circ P \sqsubseteq P$
`DataProperty:` T `SubPropertyOf:` S	$T \sqsubseteq S$
`DataProperty:` T `EquivalentTo:` S	$A \equiv C$
`EquivalentProperties:` $S_1, \ldots, S_n$	$S_i \equiv S_{i+1}$ `for` $1 \leq i < n$
`DisjointProperties:` $S_1, \ldots, S_n$	$S_i \sqsubseteq \neg S_j$ `for` $1 \leq i < j \leq n$
`DataProperty:` T `Domain:` C	$\exists T.\top_{\mathbf{D}} \sqsubseteq C$
`DataProperty:` T `Range:` D	$\top \sqsubseteq \forall T.D$
`DataProperty:` T `Characteristics: Functional`	$\top \sqsubseteq \leqslant 1\, T$
`Datatype:` B `EquivalentTo:` D	$B \equiv D$
`Class:` A `HasKey:` $U_1, \ldots, U_n$	$U_1, \ldots, U_n \text{keyfor} A$
`SameIndividual:` $i_1, \ldots, i_n$	$i_i = i_{i+1}$ `for` $1 \leq i < n$
`DifferentIndividuals:` $i_1, \ldots, i_n$	$i_i \neq i_j$ `for` $1 \leq i < j \leq n$
`Individual:` i `Types:` C	$i : C$
`Individual:` i_1 `Facts:` P i_2	$\langle i_1, i_2 \rangle : P$
`Individual:` i_1 `Facts: not` P i_2	$i_1 : (\neg \exists P.\{i_2\})$
`Individual:` i `Facts:` T v	$\langle i, v \rangle : T$
`Individual:` i `Facts: not` T v	$i : (\neg \exists T.\{v\})$

Fig. 6. OWL 2 DL Axioms and Facts

```
Facts: ex:hasBorderWith USA,
       ex:hasPopulation 33487208,
       ex:hasLandArea 9093507
```

As well as asserting one class to be a subclass of or equivalent to another, OWL 2 also allows for a set of classes to be asserted to be either equivalent or pairwise disjoint. Moreover, a class can be asserted to be equivalent to the disjoint union of a set of classes. As can be seen in Figure 6, all of these are "syntactic sugar"—they could be replaced with suitable sets of subclass or equivalence axioms.

As well as asserting one object property to be a subclass of or equivalent to another, and for equivalence and disjointness to be asserted for sets of object properties, OWL 2 also allows for a range of characteristics to be asserted for a given object property. These include asserting that the property is functional, inverse functional, reflexive, irreflexive, symmetric asymmetric and/or transitive. Moreover, the inverse of a property can be given, as well as its range and domain.

For data properties the range of available axioms is reduced: there is no inverse axiom, and the only characteristic that can be asserted is functionality. These restrictions are necessary in order to maintain a strict separation between reasoning about classes/individuals and reasoning about data ranges/values. Such a separation has been shown to allow for relatively simple integration of DL reasoners with datatype reasoners, where the datatype reasoner is used by the DL reasoner as an oracle able to answer relatively simple questions about data ranges and values [42, 49].

OWL 2 also allows new datatypes to be introduced as abbreviations for data ranges, a convenient feature if identical data ranges are used in many places in an ontology. For example, the following axiom introduces the datatype `over18` and defines it to be equivalent to the integers greater than 18:

```
Datatype: ex:over18 EquivalentTo: integer [ > 18 ]
```

One of the new features of OWL 2 is keys, and these can be introduced using a suitable axiom. For example, the following axiom states that the combination of nationality and passport number is a key for persons:

```
Class: ex:Person HasKey: ex:hasNationality, ex:hasPassportNumber
```

where `ex:hasNationality` and `ex:hasPassportNumber` are data properties. This means that no two *named* individuals can have the same nationality and passport number.

Finally, OWL 2 allows for sets of individuals to be asserted to be the same (different names for the same object) or pairwise different (no two individuals name the same object), for individuals to be asserted to be instances of one or more classes, and for both positive and negative assertions about relationships between pairs of individuals. The first two statements are provided because OWL does not make any unique name assumption (UNA), i.e., it is perfectly possible for `ex:USA` and `ex:UnitedStatesOfAmerica` to be two different names for the same object; it is therefore useful to be able to assert that a given set of names

all refer to the same object, or to assert that UNA does apply to a given set of names, i.e., that no two names from the set refer to the same object. For example, the following axioms could be used to assert the above mentioned equivalence between `ex:USA` and `ex:UnitedStatesOfAmerica`, and to assert that `ex:Alabama`, ..., `ex:Wyoming` all refer to different objects.

```
SameIndividual: USA, UnitedStatesOfAmerica
DifferentIndividuals: ex:Alabama, ..., ex:Wyoming
```

3.6 Annotations

Annotations are a mechanism for adding extra-logical "comments" to the ontology, i.e., information that does not affect the formal meaning of the ontology and can thus be ignored by a reasoning system. Annotations could include, e.g., human readable labels, provenance or hints on how the ontology should be displayed in a visualisation tool. They are sometimes also used to capture information used in language extensions, e.g., to associate a probability with an axiom in a probabilistic extension of OWL.

In OWL 2 it is possible to add annotations to almost any part of the ontology: they can be attached to the ontology itself, to entities such as classes, properties and individuals, to class and property expressions, and to statements such as axioms and declarations—they can even be attached to other annotations. Annotations are specified by using an annotation property to associate the subject of the annotation with an annotation value which can be an IRI (which could, e.g., be a class or individual name), a string literal or an anonymous individual. OWL 2 provides a number of predifined annotation properties: `rdfs:label`, `rdfs:comment`, `rdfs:seeAlso`, `rdfs:isDefinedBy`, `owl:deprecated`, `owl:versionInfo`, `owl:priorVersion`, `owl:backwardCompatibleWith` and `owl:incompatibleWith`. In addition, some simple structuring of annotation types is provided for by allowing range, domain and sub-property to be asserted for annotation properties.

This represents a significant advance over OWL, where only ontologies and entities could be annotated. In OWL this restriction was imposed due to the difficulty of representing annotated statements in RDF, the problem being that RDF has no mechanism for using a triple or a set of triples as the subject of another triple. In OWL 2 this problem has been overcome by using a form of reification for annotated statements when rendered in the RDF syntax.

Some of the uses of annotation are illustrated in the following two examples. The first example illustrates an entity (the individual `ex:Canada`) annotated with a textual comment, and the comment itself being annotated with provenance information stating that the source is the CIA Wold Fact Book:

```
Individual: ex:Canada
Annotations:
  Annotation: ex:source ex:CIA-World-Fact-Book
  rdfs:comment
```

```
"Situated in Northern North America, bordering
the North Atlantic Ocean on the east, North
Pacific Ocean on the west, and the Arctic Ocean
on the north, north of the conterminous US"
```

The second example illustrates an axiom being annotated with provenance information (again the CIA Wold Fact Book) and with information about when the axiom was last updated:

```
Individual: ex:USA
Annotation: rdfs:label ‘‘United States of America’’
Facts:
  Annotation: ex:source ex:CIA-World-Fact-Book,
              ex:lastUpdated ‘‘July 2009’’
  ex:population 307212123,
```

3.7 Global Restrictions

In order to ensure that OWL 2 is a decidable language it is necessary to impose some global restrictions on the structure of ontologies. These restrictions correspond closely to those used for the same purpose in the definition of $\mathcal{SROIQ}(\mathbf{D})$ knowledge bases. The restrictions are called global because they depend on the ontology as a whole and not just on a single expression or axiom; for example, several of the restrictions relate to the property hierarchy, which depends on the set of property axioms occurring in the imports closure of the ontology. The definitions of some of the global restrictions are rather technical, and will only be sketched here; full details can be found in Section 11 of the OWL 2 Structural Specification and Functional-Style Syntax [59].

Firstly, it is necessary to distinguish *simple* properties. Roughly speaking, a property P is not simple if its existence is implied by a chain of other properties. This is the case if, for example, P is transitive, or if P has a sub-property S, and S is transitive. In the latter case, given individuals x, y and z such that xSy and ySz, then from the transitivity of S it is possible to infer xSz, and from the fact that S is a sub-property of P it is possible to infer xPz. Intuitively, checking cardinality constraints for a non-simple property P is much more difficult because it is necessary to count not only directly related individuals but also those related via some chain of properties that implies P; in fact this leads to undecidability in $\mathcal{SROIQ}(\mathbf{D})$. Therefore, only simple roles can be used in cardinality restrictions. For similar reasons, only simple roles can be used in `self` restrictions, and in Functional, InverseFunctional, Irreflexive, Asymmetric, and Disjoint property axioms.

Secondly, the structure of property chains is restricted in various ways. They must, for example, satisfy an acyclicity condition, which is again needed in order to ensure decidability.

Thirdly, various restrictions are placed on the use of data ranges and datatype definition axioms. In particular, datatype definitions must be unique and acyclic; that is, given an axiom of the form:

`Datatype:` B `EquivalentTo:` D

where B is a newly defined datatype and D is a data range, D must not use B either directly or indirectly. This restriction means that datatype definitions can be treated as macros and simply "unfolded" by recursively substituting every occurrence of a defined datatype with the data range used to define it. In Description Logics similar restrictions on concept definitions result in an *unfoldable* (sometimes called *definitorial*) TBox—one that can be eliminated by unfolding definitions into the ABox [1].

Fourthly, the use of anonymous individuals in axioms is restricted: they are not allowed to occur at all in SameIndividual or DifferentIndividuals axioms, or in negated individual facts, and their use in other axioms must satisfy another form of acyclicity constraint. These restrictions are designed to ensure that anonymous individuals can be eliminated from the ontology using a "rolling up" technique similar to the one use in conjunctive query answering [18].

Finally, the IRIs used to name ontologies and entities in OWL 2 must not be from the *reserved vocabulary*, i.e., they must not be one of the IRIs used by the language itself. These include all of the IRIs with prefixes `rdf:`, `rdfs:`, `xsd:` and `owl:`.

4 Semantics for OWL 2

In common with Description Logics, OWL 2 has a (First Order) model-theoretic semantics called the OWL 2 Direct Semantics [53]. This semantics is basically equivalent to simply translating the ontology into a $\mathcal{SROIQ}(\mathbf{D})$ knowledge base as described in Section 3 and then applying the standard Description Logic semantics.

This model-theoretic semantics is the ultimate arbiter of the meaning of OWL 2 constructs. However, it is generally sufficient to understand the informal meaning as described above, and as described in OWL 2 user facing documents such as the Primer [57]. For example, an individual is an instance of the intersection C `and` D exactly when it is an instance of both C and D, it is an instance of a restriction P `some` C exactly when it is related via the property P to an instance of C, and it is an instance of a restriction U `value` v exactly when it is related via the data property U to the value v. Similarly, an axiom `Class:` A `SubClassOf:` C implies that every instance of A is also an instance of C, while `Class:` A `EquivalentTo:` C additionally implies that every instance of C is an instance of D.

OWL 2 includes datatypes, and these are integrated into the language in much the same way is in Description Logics that include datatypes, in particular $\mathcal{SHOQ}(\mathcal{D})$. However, the particular datatypes used in OWL 2 are taken from RDF and XML Schema Datatypes [9], and inherit the semantics from the relevant specifications. Data types, such as `xsd:integer` and data values such as `"44"^^xsd:integer` are thus given the same meaning as they have in XML Schema. For example, the interpretation domains of `xsd:integer` and

`xsd:float` are disjoint, so in OWL 2 `"1"^^xsd:integer` and `"1"^^xsd:float` are interpreted as two different values.

4.1 OWL 2 RDF-Based Semantics

For ontologies that use the RDF syntax, an alternative semantic account can be given by extending the RDF model theory with new conditions that capture the meaning of the OWL 2 vocabulary as described in the OWL 2 RDF-Based semantics [54]. For example, if an OWL 2 ontology in RDF syntax includes the triple:

< C `owl:complementOf` D >

for C and D classes, then $\mathsf{ICEXT}(C) = \mathsf{IR}\,\mathsf{ICEXT}(D)$, that is, the individuals that are instances of C ($\mathsf{ICEXT}(C)$) must be equal to the whole of the interpretation domain (IR) minus the individuals that are instances of D ($\mathsf{ICEXT}(D)$). Note the similarity to the semantics of negation in Description Logic presented in Figure 1; in fact the RDF-Based semantics is equipped with a correspondence theory that precisely defines the relationship between the two semantics[54].

In practice, the main difference between the Direct semantics and the RDF-Based semantics is that the latter can be applied to RDF-Graphs that do not respect the various restrictions on OWL 2 syntax described in Section 3; indeed, the RDF-Based semantics can be applied to arbitrary RDF graphs. It is important to be aware, however, that additional meaning (beyond that derived from the RDF semantics [24]) is only given to those parts of the graph that correspond to OWL 2 constructions as described in Section 3. It therefore makes more sense to think of the RDF-Based semantics as a semantics for OWL 2 ontologies that do not respect the restrictions required by OWL 2 DL.

This style of usage of OWL 2 is known as OWL 2 Full. Although this easing of language restrictions in OWL 2 Full can sometimes be convenient, it also has the effect of making the language undecidable [33, 47]; this makes it difficult to provide reasoning systems, and impossible to provide systems that can guarantee to correctly answer arbitrary queries. In the case of OWL Full, this resulted in applications typically using an ad hoc subset of the language along with some simple inference rules to provide reasoning that was sound but incomplete, an approach that was undesirable both from the point of view of reliability and interoperability. In OWL 2, the OWL 2 RL profile is designed in part to overcome this problem by defining a suitable subset, providing an axiomatisation that can be used as the basis for a rule set, and describing conditions on the ontology under which an implementation based on such a rule set would be sound and complete. It is conjectured that OWL 2 Full will in effect be OWL 2 RL under the RDF-Based semantics.

It is also worth pointing out that the new features of OWL 2 make it much easier for applications to live with the restrictions imposed in OWL 2 DL. In particular, punning in OWL 2 means that class/individual metamodelling is now possible in OWL 2 DL, and keys can be used instead of applying inverse

functionality to data properties—another common reason for ontologies to be OWL Full.

5 OWL 2 Profiles

As noted above, OWL 2 DL is closely related to $\mathcal{SROIQ}(\mathbf{D})$, a very expressive Description Logic. $\mathcal{SHOIN}(\mathbf{D})$ is (potentially) very difficult to reason with, as standard inference problems, such as Ontology Consistency, Class Expression Satisfiability, Class Expression Subsumption, and Instance Checking, all have 2NExpTime complexity.

For these reasons, three profiles have been defined: language fragments that have desirable computational properties, and in particular lower worst case complexities for the above mentioned inference problems (see `http://www.w3.org/TR/owl2-profiles/`). These profiles are called OWL 2 EL, OWL 2 QL and OWL 2 RL. Note that the old OWL Lite profile has been depricated as it provides no significant computational advantage (OWL Lite ontologies would, in general, be considered standard OWL 2 ontologies).

5.1 OWL 2 EL

OWL 2 EL is based on the $\mathcal{EL}^{++}$, a Description Logic for which standard reasoning problems can be performed in time that is polynomial with respect to the size of the ontology. This profile captures the expressive power used by many ontologies used in the life sciences, and is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes, as is often the case with life science ontologies.

Unlike the other two profiles, EL has the advantage of being "symmetrical", in the sense that the restrictions apply equally to class expressions occurring on the left hand and the right hand side of class inclusion axioms, making it very easy to define and to use. The restrictions on class expressions rule out the use of universal quantification (i.e., ObjectAllValuesFrom and DataAllValuesFrom), cardinality restrictions, disjunction, negation, enumerations involving multiple individuals (i.e., ObjectOneOf and DataOneOf), and most property characteristics (including InverseObjectProperties, DisjointObjectProperties, DisjointDataProperties, IrreflexiveObjectProperty, FunctionalObjectProperty, InverseFunctionalObjectProperty, SymmetricObjectProperty and AsymmetricObjectProperty. In addition the set of supported datatypes has been designed such that the intersection of the value spaces of any set of datatypes is either empty or infinite, which is necessary to retain the desired computational properties. As a result, use of the following datatypes is ruled out: 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, EL also rules out the use of anonymous individuals.

Several reasoners are available for OWL 2 EL, including CB [36], CEL [4], Pellet (in fact Pellet supports all of OWL 2, but includes a dedicated EL reasoner for optimised reasoning with this profile) [74] and Snorocket [39, 14]. These reasoners all use a saturation based technique in which the TBox extended so as to explicitly include all subsumption relationships holding between named classes. These algorithms are extremely effective at dealing with large ontologies: the CB reasoner can, for example, fully classify the 400,000 class SNOMED-CT ontology [76] in less than 60 seconds. Recent work has shown how scalability can also be extended to large data sets by using database technology to store the set of individual axioms (the ABox) and employing a mixture of materialisation and query rewriting [44].

5.2 OWL 2 QL

OWL 2 QL is based on DL-Lite$_R$, a Description Logic for which conjunctive query answering can be implemented using conventional relational database systems and so can be performed in LOGSPACE with respect to the size of the data (individual axioms). It is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. 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 OWL 2 QL is necessarily quite limited, although it does include most of the main features of conceptual models such as UML class diagrams and ER diagrams.

Several variants of DL-Lite have been described in the literature, with OWL 2 QL being based on the DL-Lite$_R$ variant. This has the advantage that, although the instance data is assumed to be in a relational database, no unique name assumption (UNA) is required—this is because the UNA would have no impact on the semantic consequences of a DL-Lite$_R$ ontology. OWL 2 QL not only restricts the kinds of class expression that can be used, but also varies these restrictions depending where the expression occurs in an axiom (e.g., as the subclass or superclass part of a `subClassOf` axiom. This makes the precise definition of the profile rather complex, and the reader is referred to the OWL 2 Profiles specification for full details [58]. The set of supported datatypes is the same as for OWL 2 EL, and the use of anonymous individuals is similarly ruled out. Finally, OWL 2 QL does not support individual equality assertions (SameIndividual), because this would make queries no longer first order rewritable, with the result that query answering could no longer be implemented directly using relational database technologies. However, individual equality could be materializing the equality relation in the database and using the resulting relation in query answering [69].

Several reasoners are available for DL-Lite$_R$/OWL 2 QL, including Owlgres (see `http://pellet.owldl.com/owlgres`), and QuOnto (see `http://www.dis.uniroma1.it/~quonto/`). Both of these are based on query rewriting techniques that transform a conjunctive query against the ontology into a set of queries against the individual axioms (the ABox) only, and ultimately (via mappings

from ontology class and property names to SQL queries) into SQL queries against a database (or databases) where the instance data is stored [65].

5.3 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, and presenting a partial axiomatization of the 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 4.1). The design of OWL 2 RL was inspired by Description Logic Programs [20] and pD* [79].

Like OWL 2 QL, the syntax of RL is asymmetric in the sense that different constraints apply to class expressions depending where they occur in ontology axioms. Essentially, this means allowing enumerations (ObjectOneOf), intersections (ObjectIntersectionOf), unions (ObjectUnionOf) and existential restrictions in the subclass position of a `subClassOf` axiom, and intersection (ObjectIntersectionOf), negation (ObjectComplementOf), universal restrictions (ObjectAllValuesFrom and DataAllValuesFrom), existential restrictions using an individual or data value (ObjectHasValue and DataHasValue) and at-most 0/1 cardinality restrictions (ObjectMaxCardinality and DataMaxCardinality with values 0 or 1) in the superclass position of a `subClassOf` axiom (see the OWL 2 Profiles specification for full details [58]).

For ontologies satisfying the above mentioned syntactic constraints, a suitable rule-based implementation (e.g., one based on the partial axiomatization of the RDF-Based semantics) will have desirable computational properties; for example, it can return all and only the correct answers to ground atomic queries (see Theorem PR1 from the OWL 2 Profiles specification [58] and the OWL 2 Conformance specification [52]). As mentioned in Section 4.1, such an implementation can also be used with arbitrary RDF graphs. In this case, however, the above mentioned computational 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.

Several reasoners are available for OWL 2 RL, including Elly (see `http://elly.sourceforge.net/`), Jena (see `http://jena.sourceforge.net/`) and the Oracle Database 11g OWL Reasoner (see `http://www.oracle.com/technology/tech/semantic_technologies/index.html`). These implementations are all based on rule extended triple stores and relational databases and work by computing all "relevant" inferences and materialising them in the store/database. This may require significant additional time and storage, but if this is within acceptable bounds, then queries can subsequently be answered simply by querying the store/database. Completeness is, however, dependent on the set of materialised inferences and the kind of query being answered and,

as mentioned above, can only be guaranteed for ground atomic queries against ontologies satisfying the syntactic restrictions on the RL profile.

6 OWL Tools and Applications

As discussed in Section 2, one of the motivations for basing OWL on a description logic was the ready availability of tools and infrastructure. Similarly, in the case of OWL 2, the willingness and ability of tool developers to support the language was an important influence on its design.

Regardless of the target ontology language, developing good quality ontologies is an extremely difficult and time-consuming task. It is therefore essential to provide ontology engineers with tool support. A range of ontology development tools are available for this purpose, including Swoop [34], Protégé [38], and TopBraid Composer (see `http://www.topbraidcomposer.com/`).

Ontology development tools invariably use a DL reasoner to provide feedback to the user about the logical implications of their design, e.g., to warn them about inconsistencies. Moreover, reasoners are an essential feature of applications where they are required, e.g., in order to answer both conceptual and data retrieval queries.

The availability of tools and reasoning systems has been an important factor in the increasingly widespread use of OWL. Applications of OWL are particularly prevalent in the life sciences where it has been used by the developers of several large biomedical ontologies, including the SNOMED, GO and BioPAX ontologies mentioned above, the Microarray Gene Expression Data (MGED) ontology (see `http://mged.sourceforge.net/ontologies/index.php`), the Foundational Model of Anatomy (FMA) [19] and the National Cancer Institute thesaurus (NCI) [23].

Another important component in OWL applications, including the above mentioned editors, is the de facto standard Manchester OWL API [25]. The API takes care of parsing and rendering OWL ontologies in various different syntaxes, and also provides a standard interface to OWL reasoners. This means that OWL applications can easily switch between reasoners, choosing whichever proves to be most effective.

6.1 Ontology Design Tools

Ontologies are often large and complex: the well known SNOMED clinical terms ontology includes, for example, more than 400,000 class names [75]. Building and maintaining such ontologies is very costly and time consuming, and providing tools and services to support this "ontology engineering" process is of crucial importance to both the cost and the quality of the resulting ontology. State of the art ontology development tools, such as Swoop [34], Protégé [38], and TopBraid Composer (see `http://www.topbraidcomposer.com/`), use a DL reasoner, such as FaCT++ [80], HermiT [51], Racer [22] or Pellet [74], to provide feedback to the user about the logical implications of their design.

The importance of reasoning support in ontology applications was highlighted in a recent paper describing a project in which the Medical Entities Dictionary (MED), a large ontology (100,210 classes and 261 properties) that is used at the Columbia Presbyterian Medical Center, was converted into OWL and checked using an OWL reasoner [37]. This check revealed "systematic modelling errors", and a significant number of missed subClassOf relationships which, if not corrected, "could have cost the hospital many missing results in various decision support and infection control systems that routinely use MED to screen patients".

Similarly, an extended version of the SNOMED ontology was checked using an OWL reasoner, and a number of missing subClassOf relationships found. This ontology is being used by the UK National Health Service (NHS) to provide "A single and comprehensive system of terms, centrally maintained and updated for use in all NHS organisations and in research", and as a key component of their multi-billion pound "Connecting for Health" IT programme. An important feature of this system is that it can be extended to provide more detailed coverage if needed by specialised applications. For example, a specialist allergy clinic may need to distinguish allergies caused by different kinds of nut, and so may add new terms to the ontology such as AlmondAllergy:

$$\texttt{AlmondAllergy} \equiv \texttt{Allergy} \sqcap \exists \texttt{causedBy.Almond}$$

Using a reasoner to insert this new term into the ontology will ensure that it is recognised as a subClassOf NutAllergy. This is clearly of crucial importance in order to ensure that patients with an AlmondAllergy are correctly identified in the national records system as patients having a NutAllergy.

As well as computing the class hierarchy, ontology design tools typically provide (at least) warnings about inconsistencies and redundancies. An inconsistent (sometimes called unsatisfiable) class is one whose description is "overconstrained", with the result that it can never have any instances. This is typically an unintended feature of the design—why introduce a name for a class that can never have any instances—and may be due to subtle interactions between axioms. It is, therefore, very useful to be able to detect such classes and bring them to the attention of the ontology engineer. For example, during the development of an OWL ontology at the NASA Jet Propulsion Laboratory (see Section 6.2), the class "OceanCrustLayer" was found to be inconsistent. This was discovered (with the help of debugging tools) to be the result of its being defined to be both a region and a layer, one of which (layer) was a 2-dimensional object and the other a 3-dimensional object, where the axioms describing 2-dimensional and 3-dimensional objects ensured that these two classes were disjoint (had no instances in common). The inconsistency thus highlighted a fundamental error in the design of the ontology, discovering and repairing which obviously improved the quality of the ontology.

It is also possible that the descriptions in the ontology mean that two classes necessarily have exactly the same set of instances, i.e., that they are alternative names for the same class. This may be desirable in some situations, e.g., to

capture the fact that "Myocardial infarction" and "Heart attack" mean the same thing. It could, however, also be the inadvertent result of interactions between descriptions, and so it is also useful to be able to alert users to the presence of such "synonyms". For example, when developing a medical terminology ontology a domain expert added the following two axioms:

$$\texttt{AspirinTablet} \equiv \exists \texttt{hasForm.Tablet}$$
$$\texttt{AspirinTablet} \sqsubseteq \texttt{AspirinDrug}$$

intending to capture the information that aspirin tablets are just those aspirin drugs that have the form of a tablet. Instead, these axioms had the effect of making *every* kind of tablet be an aspirin tablet. This was immediately corrected when the reasoner alerted the domain expert to the unexpected equivalence between `Tablet` and `AsprinTablet`.

As discussed above, ontology development tools usually check for implicit subsumption relationships, and update the class hierarchy accordingly. This is also a very useful design aid: it allows ontology developers to focus on class descriptions, leaving the computation of the class hierarchy to the reasoner, and it can also be used by developers to check if the hierarchy induced by the class descriptions is consistent with their intuition. This may not be the case when, for example, errors in the ontology result in unexpected subsumption inferences, or "under-constrained" class descriptions result in expected inferences not being found. The latter case is extremely common, as it is easy to inadvertently omit axioms that express "obvious" information. For example, an ontology engineer may expect the class of patients who have a fracture of both the tibia and the fibula to be a subClassOf "patient with multiple fractures"; however, this may not be the case if the ontology doesn't include (explicitly or implicitly) the information that the tibia and fibula are different bones. Failure to find the expected subsumption relationship will alert the engineer to the missing DisjointClasses axiom.

Recent work has also shown how reasoning can be used to support modular design [16] and module extraction [15], important techniques for working with large ontologies. When developing a large ontology such as SNOMED, it is useful if not essential to divide the ontology into modules, e.g., to facilitate parallel work by a team of ontology developers. Reasoning techniques can be used to alert the developers to unanticipated and/or undesirable interactions between the various modules. Similarly, it may be desirable to extract from a large ontology a smaller module containing all the information relevant to some subset of the domain, e.g., heart disease—the resulting small(er) ontology will be easier for humans to understand and easier for applications to use. Reasoning can be used to compute a module that is as small as possible while still containing all the necessary information.

Finally, in order to maximise the benefit of reasoning services, tools should be able to explain inferences: without this facility, users may find it difficult to repair errors in the ontology and may even start to doubt the correctness of inferences. Explanation typically involves computing a (hopefully small) subset of

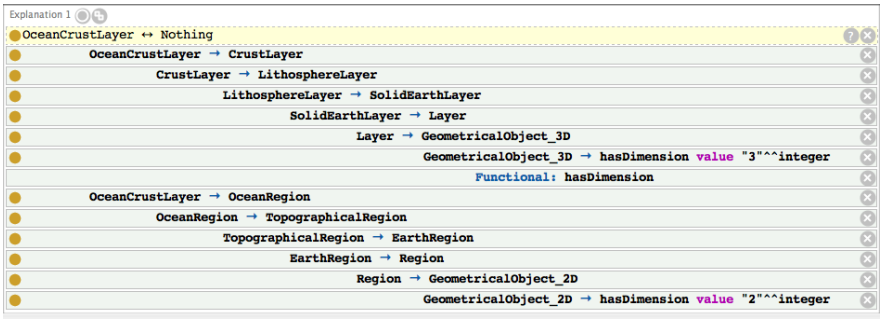

Fig. 7. An explanation from PROTÉGÉ

the ontology that still entails the inference in question, and if necessary presenting the user with a chain of reasoning steps [35]. Figure 7, for example, shows an explanation, produced by the PROTÉGÉ ontology development tool, of the above mentioned inference with respect to the inconsistency of OceanCrustLayer.

6.2 Reasoning in Deployed Applications

Reasoning is also important when ontologies are deployed in applications—it is needed, e.g., in order to answer structural queries about the domain and to retrieve data. For example, biologists use ontologies such as the Gene Ontology (GO) and the Biological Pathways Exchange ontology (BioPAX) to annotate data from gene sequencing experiments so as to be able to answer complex queries such as "what DNA binding products interact with insulin receptors". Answering this query requires a reasoner not only to identify individuals that are (perhaps only implicitly) instances of DNA binding products and of insulin receptors, but also to identify which pairs of individuals are (perhaps only implicitly) related via the interactsWith property.

It is easy to imagine that, with large ontologies, query answering may be a very complex task. The use of DL reasoners allows OWL ontology applications to answer complex queries, and to provide guarantees about the correctness of the result. This is obviously of crucial importance when ontologies are used in safety critical applications such as medicine; it is, however, also important if ontology based systems are to be used as components in larger applications, such as the Semantic Web, where the correct functioning of automated processes may depend on their being able to (correctly) answer such queries.

Ontologies are also widely used to facilitate the sharing and integration of information. The Neurocommons project (see `http://sciencecommons.org/projects/data/`) for example, aims to provide a platform for sharing and integrating knowledge in the neuroscience domain. A key component is an ontology of annotations that will be used to integrate available knowledge on the web, including major neuroscience databases. Similarly, the OBO Foundry (see

http://www.obofoundry.org/) is a library of ontologies designed to facilitate information sharing and integration in the biomedical domain, while the NCI ontology mentioned above constitutes the working vocabulary used in NCI data systems (see http://ncicb.nci.nih.gov/NCICB/core/EVS/).

In information integration applications the ontology can play several roles: it can provide a formally defined and extensible vocabulary for use in semantic annotations, it can be used to describe the structure of existing sources and the information that they store, and it can provide a detailed model of the domain against which queries can be formulated. Such queries can be answered by using semantic annotations and structural knowledge to retrieve and combine information from multiple sources [78]. It should be noted that the use of ontologies in information integration is far from new, and has already been the subject of extensive research within the database community [5].

Other examples of OWL ontology applications include:

- United Nations Food and Agriculture Organization (FAO) is using OWL to develop a range of ontologies covering areas such as agriculture and fisheries (see http://www.fao.org/agris/aos/Applications/intro.htm).
- The Semantic Web for Earth and Environmental Terminology (SWEET) ontologies developed at the US National Aeronautics and Space Administration (NASA) Jet Propulsion Laboratory (see http://sweet.jpl.nasa.gov/ontology/). These include ontologies describing space, the biosphere and the sun. SWEET is now being expanded by a number of earth and space science efforts, and has been augmented in the GEON project (see http://www.geongrid.org/) to cover the solid earth, and by the Virtual Solar Terrestrial Observatory Project (see http://vsto.hao.ucar.edu/) to include much more information on the atmosphere.
- An ontology used at General Motors in a project to help quality improvement activities for assembly line processes in different production sites [46].

7 Summary

OWL 2 exhibits the desirable features of Description Logics, including useful expressive power, formal syntax and semantics, decidability, and practical reasoning systems, resulting in OWL 2 providing effective ontology representation facilities. As well, OWL 2 is firmly a part of the W3C Semantic Web, with its use of IRIs for names, XML Schema datatypes, and ontologies as Web documents, which can then import other OWL 2 ontologies over the Web. OWL 2 thus firmly integrates ontologies into the Semantic Web.

It is not necessary to understand all of the formal aspects of OWL 2 in order to use OWL 2 effectively. All that is required is a reasonable understanding of what it means to define aspects of a class, property, or individual in an ontology. The use of OWL 2 ontology editors, such as Protégé, serve as a bridge between the syntax of an ontology and its semantics, calling OWL 2 reasoners,

such as Pellet, HermiT, and FaCT++ to determine consequences of the ontology statements and then present them in an easy-to-see fashion.

It is also not necessary to completely understand the differences between the various profiles of OWL 2. It is possible to design an ontology without reference to the profiles, and without checking which profile the ontology belongs to. Of course, if the computational or implementation benefits of a particular profile are needed, it is best to keep the ontology being designed within that particular profile. It is expected that ontology editors will soon be able to both check which profile an ontology is in, and impose syntactic constrains to ensure that an ontology stays within a given profile.

The situation is somewhat more complex with OWL 2 Full. First, determining whether an ontology is outside of OWL 2 DL has to be done on the entire ontology. Second, most OWL 2 tools do not handle OWL 2 Full, so using OWL 2 Full results in a loss of tool support. Third, reasoning in OWL 2 Full is undecidable, so there is no chance of effective reasoning tools being developed. The use of OWL 2 Full should thus be reserved for situations where there already is some existing RDFS that does not fit within OWL 2 DL and cannot be modified.

In practice, relatively few OWL Full applications have emerged to date, and where OWL Full ontologies are found, they often turn out to be outside the OWL DL subset only as the result of minor syntactic errors and thus should have been in OWL DL. Fragments of OWL 2 are, however, sometimes used as ad hoc extensions to RDFS. A common example is the use of OWL functional properties, and explicit equivalences and (in)equalities, in what would otherwise be an RDFS ontology. In many cases, the RDFS ontology can or should be in OWL 2 DL, but sometimes there is some significant aspect of the RDFS ontology that requires the use of OWL Full. The need to use OWL Full should be further reduced in OWL 2 due to support for punning and key constraints.

There remain, of course, significant issues that are not handled by OWL 2, but that are definitely pertinant to the Semantic Web.

- OWL 2 excludes useful expressive power to remain decidable, or just because there was not enough time in the OWL Working Group to specify the construct. In particular DL-safe rules [50] and Description Graphs [48] are not in OWL 2, even though there are Description Logics that include these features and even Description Logic reasoners that implement them (such as HermiT). Similarly there are proposals for N-ary relations [13] and N-ary datatypes [60] in Description Logics but neither of these features are in OWL 2.
- OWL 2 is monotonoic, as are most formal representation languages that can handle reasonably large amounts of information, and thus cannot handle default reasoning or localised closed world assumptions.
- OWL 2 is binary, as are most formal representation languages that can handle reasonably large amounts of information, and thus cannot handle probabilistic or fuzzy reasoning.
- OWL 2 treats ontologies as single entities and does not allow the extraction of part of an ontology.

Some of the non-present features are already in particular proposals for extensions for OWL 2. If considered desirable by a sufficiently large community, it is likely that OWL 2 reasoners and other systems will implement them in a compatible fashion, leading to *de facto* extensions to OWL 2. Other non-present features, however, would require significant research to provide formal foundations, reasoning algorithms, and/or effective reasoners for them, which must be done before they can be included in future OWL 2 extensions.

References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
2. F. Baader and P. Hanschke. A schema for integrating concrete domains into concept languages. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 452–457, 1991.
3. F. Baader and B. Hollunder. $\mathcal{KRIS}$: $\mathcal{K}$nowledge $\mathcal{R}$epresentation and $\mathcal{I}$nference $\mathcal{S}$ystem. *SIGART Bull.*, 2(3):8–14, 1991.
4. F. Baader, C. Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. SV, 2006.
5. C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
6. S. Bechhofer, C. Goble, and I. Horrocks. DAML+OIL is not enough. In *Proc. of the 2001 Int. Semantic Web Working Symposium (SWWS 2001)*, pages 151–159, 2001. Available at `http://www.semanticweb.org/SWWS/program/full/SWWSProceedings.pdf`.
7. S. Bechhofer, I. Horrocks, C. Goble, and R. Stevens. OilEd: A Reason-able ontology editor for the semantic web. In *Proc. of the Joint German/Austrian Conf. on Artificial Intelligence (KI 2001)*, number 2174 in Lecture Notes in Artificial Intelligence, pages 396–408. Springer, 2001.
8. D. Beckett. Rdf/xml syntax specification (revised). W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-syntax-grammar/`.
9. P. V. Biron and A. Malhotra. XML schema part 2: Datatypes. W3C Recommendation, May 2001. Available at `http://www.w3.org/TR/xmlschema-2/`.
10. R. J. Brachman. *A Structural Paradigm for Representing Knowledge*. PhD thesis, Harvard University, Cambridge, MA, 1977. Revised version published as BBN Report No. 3605, Bolt Beranek and Newman, Inc., Cambridge, MA, July, 1978.
11. R. J. Brachman and J. G. Schmolze. An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9(2):171–216, 1985.
12. P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive description logics: Preliminary report. In *Proc. of the 1995 Description Logic Workshop (DL'95)*, pages 131–139, 1995.
13. D. Calvanese, G. De Giacomo, and M. Lenzerini. Reasoning in expressive description logics with fixpoints based on automata on infinite trees. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 84–89, 1999.

14. R. Cornet and K. A. Spackman, editors. *Proceedings of the Third International Conference on Knowledge Representation in Medicine, Phoenix, Arizona, USA, May 31st - June 2nd, 2008*, volume 410 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
15. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. of the Sixteenth International World Wide Web Conference (WWW 2007)*, 2007.
16. B. Cuenca Grau, Y. Kazakov, I. Horrocks, and U. Sattler. A logical framework for modular integration of ontologies. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 298–303, 2007.
17. D. Fensel, F. van Harmelen, I. Horrocks, D. McGuinness, and P. F. Patel-Schneider. OIL: An ontology infrastructure for the semantic web. *IEEE Intelligent Systems*, 16(2):38–45, 2001.
18. B. Glimm, I. Horrocks, C. Lutz, and U. Sattler. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of the 20th Int. Joint Conf. on Artificial Intelligence (IJCAI 2007)*, pages 399–404, 2007.
19. C. Golbreich, S. Zhang, and O. Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, 4(3):181–195, 2006.
20. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description logic programs: Combining logic programs with description logic. In *Proc. of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.
21. W. E. Grosso, H. Eriksson, R. W. Fergerson, J. H. Gennari, S. W. Tu, and M. A. Musen. Knowledge modelling at the millenium (The design and evolution of Protégé-2000). In *Proc. of Knowledge acqusition workshop (KAW'99)*, 1999.
22. V. Haarslev and R. Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.
23. F. W. Hartel, S. de Coronado, R. Dionne, G. Fragoso, and J. Golbeck. Modeling a description logic vocabulary for cancer research. *Journal of Biomedical Informatics*, 38(2):114–129, 2005.
24. P. Hayes. RDF model theory. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/rdf-mt/`.
25. M. Horridge, S. Bechhofer, and O. Noppens. Igniting the OWL 1.1 touch paper: The OWL API. In *Proc. of the Third OWL Experiences and Directions Workshop*, number 258 in CEUR (`http://ceur-ws.org/`), 2007.
26. M. Horridge, N. Drummond, J. Goodwin, A. Rector, R. Stevens, and H. Wang. The Manchester OWL syntax. In *Proc. of the Second OWL Experiences and Directions Workshop*, volume 216 of *CEUR (`http://ceur-ws.org/`)*, 2006.
27. I. Horrocks. Using an expressive description logic: FaCT or fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.
28. I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. OIL: The Ontology Inference Layer. Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences, Sept. 2000. See http://www.ontoknowledge.org/oil/.
29. I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2006)*, pages 57–67. AAAI Press, 2006.
30. I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

31. I. Horrocks and U. Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proc. of the 17th Int. Joint Conf. on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.
32. I. Horrocks and U. Sattler. A tableaux decision procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.
33. I. Horrocks, U. Sattler, and S. Tobies. Practical reasoning for expressive description logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.
34. A. Kalyanpur, B. Parsia, E. Sirin, B. Cuenca-Grau, and J. Hendler. SWOOP: a web ontology editing browser. *J. of Web Semantics*, 4(2), 2005.
35. A. Kalyanpur, B. Parsia, E. Sirin, and J. Hendler. Debugging unsatisfiable classes in OWL ontologies. *J. of Web Semantics*, 3(4):243–366, 2005.
36. Y. Kazakov. Consequence-driven reasoning for horn shiq ontologies. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 2040–2045, 2009.
37. A. Kershenbaum, A. Fokoue, C. Patel, C. Welty, E. Schonberg, J. Cimino, L. Ma, K. Srinivas, R. Schloss, and J. W. Murdock. A view of OWL from the field: Use cases and experiences. In *Proc. of the Second OWL Experiences and Directions Workshop*, volume 216 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2006.
38. H. Knublauch, R. Fergerson, N. Noy, and M. Musen. The Protégé OWL Plugin: An open development environment for semantic web applications. In S. A. McIlraith, D. Plexousakis, and F. van Harmelen, editors, *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 229–243. Springer, 2004.
39. M. Lawley. Exploiting fast classification of snomed ct for query and integration of health data. In Cornet and Spackman [14].
40. H. J. Levesque and R. J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.
41. C. Lutz. Reasoning with concrete domains. In *Proc. of the 16th Int. Joint Conf. on Artificial Intelligence (IJCAI'99)*, pages 90–95, 1999.
42. C. Lutz. *The Complexity of Reasoning with Concrete Domains*. PhD thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.
43. C. Lutz, C. Areces, I. Horrocks, and U. Sattler. Keys, nominals, and concrete domains. *J. of Artificial Intelligence Research*, 23:667–726, 2004.
44. C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic el using a relational database system. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI 2009)*, pages 2070–2075, 2009.
45. R. MacGregor. Inside the LOOM description classifier. *SIGART Bull.*, 2(3):88–92, 1991.
46. A. P. Morgan, J. A. Cafeo, K. Godden, R. M. Lesperance, A. M. Simon, D. L. McGuinness, and J. L. Benedict. The general motors variation-reduction adviser. *AI Magazine*, 26(2), 2005.
47. B. Motik. On the properties of metamodeling in OWL. In *Proc. of the 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 548–562. Springer, 2005.

48. B. Motik, B. Cuenca Grau, I. Horrocks, and U. Sattler. Representing structured objects using description graphs. In *Proc. of the 11th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2008)*, pages 296–306, 2008.
49. B. Motik and I. Horrocks. OWL datatypes: Design and implementation. In *Proc. of the 7th International Semantic Web Conference (ISWC 2008)*, volume 5318 of *Lecture Notes in Computer Science*, pages 307–322. Springer, 2008.
50. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. In *Proc. of the 3rd International Semantic Web Conference (ISWC 2004)*, volume 3298, pages 549–563, 2004.
51. B. Motik, R. Shearer, and I. Horrocks. Optimized reasoning in description logics using hypertableaux. In *Proc. of the 21st Int. Conf. on Automated Deduction (CADE-21)*, volume 4603 of *Lecture Notes in Artificial Intelligence*, pages 67–83. Springer, 2007.
52. OWL 2 Web Ontology Language Conformance. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-conformance/`.
53. OWL 2 Web Ontology Language Direct Semantics. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-direct-semantics/`.
54. OWL 2 Web Ontology Language RDF-Based Semantics. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-rdf-based-semantics/`.
55. OWL 2 Web Ontology Language Manchester Syntax. W3C Working Draft, 11 June 2009. Available at `http://www.w3.org/TR/owl2-manchester-syntax/`.
56. OWL 2 Web Ontology Language Overview. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-overview/`.
57. OWL 2 Web Ontology Language Primer. W3C Working Draft, 11 June 2009. Available at `http://www.w3.org/TR/owl2-primer/`.
58. OWL 2 Web Ontology Language Profiles. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-profiles/`.
59. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax. W3C Candidate Recommendation, 11 June 2009. Available at `http://www.w3.org/TR/owl2-syntax/`.
60. B. Parsia and U. Sattler. OWL 2 Web Ontology Language Data Range Extension: Linear Equations. W3C Working Draft, 11 June 2009. Available at `http://www.w3.org/TR/owl2-dr-linear/`.
61. P. F. Patel-Schneider. Small can be beautiful in knowledge representation. In *Proc. of the IEEE Workshop on Knowledge-Based Systems*, 1984. An extended version appeared as Fairchild Tech. Rep. 660 and FLAIR Tech. Rep. 37, October 1984.
62. P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web Ontology Language semantics and abstract syntax. W3C Recommendation, 10 February 2004. Available at `http://www.w3.org/TR/owl-semantics/`.
63. P. F. Patel-Schneider, D. L. McGuiness, R. J. Brachman, L. A. Resnick, and A. Borgida. The CLASSIC knowledge representation system: Guiding principles and implementation rational. *SIGART Bull.*, 2(3):108–113, 1991.
64. Pellet OWL reasoner. Maryland Information and Network Dynamics Lab, 2003. `http://www.mindswap.org/2003/pellet/index.shtml`.
65. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, 10:133–173, 2008.
66. RFC 3987: Internationalized Resource Identifiers (IRIs). Internet Engineering Task Force (IETF) Request For Comments (RFC), January 2005. Available at `http://www.ietf.org/rfc/rfc3987.txt`.

67. RIF RDF and OWL Compatibility. W3C Recommendation, 22 June 2010. Available at `http://www.w3.org/TR/rif-rdf-owl/`.
68. R. B. Roberts and I. P. Goldstein. The FRL primer. Memo 408, Massachusetts Institute of Technology Artificial Intelligence Laboratory, July 1977. Available at `ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-408.pdf`.
69. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier, 2001.
70. U. Sattler. A concept language extended with different kinds of transitive roles. In G. Görz and S. Hölldobler, editors, *Proc. of the 20th German Annual Conf. on Artificial Intelligence (KI'96)*, number 1137 in Lecture Notes in Artificial Intelligence, pages 333–345. Springer, 1996.
71. K. Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI'91)*, pages 466–471, 1991.
72. M. Schmidt-Schauß. Subsumption in KL-ONE is undecidable. In R. J. Brachman, H. J. Levesque, and R. Reiter, editors, *Proc. of the 1st Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'89)*, pages 421–431. Morgan Kaufmann, Los Altos, 1989.
73. M. Schmidt-Schauß and G. Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.
74. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. of Web Semantics*, 5(2):51–53, 2007.
75. K. Spackman. Managing clinical terminology hierarchies using algorithmic calculation of subsumption: Experience with SNOMED-RT. *J. of the Amer. Med. Informatics Ass.*, 2000. Fall Symposium Special Issue.
76. K. Spackman, K. Campbell, and R. Cote. SNOMED RT: A reference terminology for health care. *J. of the American Medical Informatics Association*, pages 640–644, 1997. Fall Symposium Supplement.
77. SPARQL query language for RDF. W3C Recommendation, 15 January 2008. Available at `http://www.w3.org/TR/rdf-sparql-query/`.
78. R. Stevens, P. Baker, S. Bechhofer, G. Ng, A. Jacoby, N. W. Paton, C. A. Goble, and A. Brass. Tambis: Transparent access to multiple bioinformatics information sources. *Bioinformatics*, 16(2):184–186, 2000.
79. H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *J. of Web Semantics*, 3(2–3):79–115, 2005.
80. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.