8

Ontology Languages and Applications

As discussed in Section 1.2, DL systems have been used in a range of application domains, including configuration, software information and documentation systems and databases, where they have been used to support schema design, schema and data integration, and query answering. More recently, DLs have played a central role in the semantic web [Hor08], having been adopted as the basis for ontology languages such as OIL, DAML+OIL and OWL [HPSvH03]. This has rapidly become the most prominent application of DLs, and DL knowledge bases are now often referred to as ontologies.

In computer science, an ontology is a conceptual model specified using some ontology language; this idea was succinctly captured by Gruber in his definition of an ontology as "an explicit specification of a conceptualisation" [Gru93]. Early ontology languages were often based on frames, but as in the case of early DLs, a desire to provide them with precise semantics and well-defined reasoning procedures increasingly led to ontology languages becoming logic-based. The OIL ontology language was something of a compromise: it had a frame-based syntax, but complemented this with a formal semantics based on a mapping to $\mathcal{SHIQ}$. In DAML+OIL and OWL the DL-based semantics were retained, but the frame-based syntax of OIL was replaced with a structure much closer to DL-style axioms.

In Section 8.1 we will discuss OWL in more detail, examining its relationship to RDF and to $\mathcal{SROIQ}$, its syntax (or rather syntaxes), some features that go beyond what is typically found in a DL, and its various profiles or sub-languages. In Section 8.2 we will look at some interesting examples of OWL tools and applications.

8.1 The OWL ontology language

OWL is a semantic web ontology language developed by the World Wide Web Consortium (W3C), an international community that defines Web technologies. W3C follows a consensus-driven process for the publication of specification documents for Web technologies, in particular *Recommendations*, which are considered Web standards. OWL was first standardised in 2004, and then revised in 2012, with the revision being denoted OWL 2. Although using a variety of more "Web-friendly" syntaxes based, e.g., on XML and RDF, the basic structure of OWL corresponds closely with that of a DL, and includes such familiar constructs as existential and value restrictions, (qualified) number restrictions, inverse roles, nominals and role hierarchies (see Chapter 2). Moreover, the semantics of OWL can be defined via a mapping into an expressive DL.[1]

8.1.1 OWL and RDF

OWL was designed to extend RDF, a pre-existing W3C Recommendation. It is beyond the scope of this chapter to provide a detailed description of RDF, but a brief sketch will be useful in order to explain some of the features of OWL; interested readers are referred to `http://www.w3.org/RDF/` for complete information. RDF provides a very simple graph-like data model, with each statement, or *triple*, representing a labelled, directed edge in the graph. A triple consists of three elements called the subject, predicate and object, and they are often written

$$\langle s, p, o \rangle$$

where s is the subject, p the predicate and o the object. Such a triple represents a p-labelled edge from vertex s to vertex o; it can also be thought of as a first-order logic ground atomic formula $p(s, o)$, where p is a binary predicate and s and o are constants.

In RDF, all subject, predicate and object names are Internationalized Resource Identifiers (IRIs) [RFC05], a generalised version of the URLs that are used to identify resources on the Web. As they can be rather verbose, IRIs are often abbreviated by defining one or more common prefixes for the IRIs used in an ontology, e.g., by writing the IRI *http://dl.book/example#name* as *eg:name*, where *eg:* is defined to be the

[1] Roughly speaking, OWL can be mapped into $\mathcal{SHOIN}$, and OWL 2 into $\mathcal{SROIQ}$.

prefix *http://dl.book/example#*. Furthermore, a default prefix is often defined for all IRIs used in a given document, so we can simply write *:name* if *eg:* is the default prefix.

RDF assigns special meanings to certain predicates. In particular, *rdf*:*type* represents the "instance of" relationship, and is used to capture unary predicate formulae, where *rdf:* is the prefix *http://www.w3.org/1999/02/22-rdf-syntax-ns#*. For example, the triple $\langle s, \mathit{rdf}{:}\mathit{type}, o\rangle$ can be thought of as representing the first-order logic ground atomic formula $o(s)$, where o is a unary predicate and s is a constant. RDF thus provides a very natural way to capture ABox assertions, with a triple $\langle a, r, b\rangle$ corresponding to a role assertion $(a, b) : r$ and $\langle a, \mathit{rdf}{:}\mathit{type}, C\rangle$ corresponding to a concept assertion $a : C$. Note that in RDF and OWL, roles are referred to as properties and concepts are referred to as classes, so $\langle a, r, b\rangle$ would be called a property assertion and $\langle a, \mathit{rdf}{:}\mathit{type}, C\rangle$ a class assertion. In RDF C is always atomic (i.e., a class name), but in OWL C could be part of a graph that defines a compound class (see the OWL syntax example below).

RDF Schema (RDFS) extends the set of special predicates in order to capture a limited set of "schema" level statements, many of which correspond to TBox axioms;[2] for example, the triple $\langle C, \mathit{rdfs}{:}\mathit{subClassOf}, D\rangle$ corresponds to a TBox axiom $C \sqsubseteq D$.

OWL further extends this set of special predicates to capture more complex concept expressions and TBox axioms. Unlike a DL knowledge base, an OWL ontology makes no distinction between TBox and ABox – it consists of a single set of RDF triples (also known as an RDF graph) representing ABox assertions and/or TBox axioms. This style of syntax based on triples is flexible, but when extended to capture complex concepts it becomes quite cumbersome, and complicates even basic tasks; for example, it is difficult to constrain the syntax so as to allow only syntactically valid axioms and assertions, and to parse documents into an internal representation of a set of axioms. OWL is therefore defined using a functional-style syntax [OWL12c], along with a bidirectional mapping between this syntax and RDF triples [OWL12a]. In addition, other syntaxes for OWL have been specified, including the Manchester syntax, which presents an ontology in a succinct form easily readable by humans. The following example gives the DL axiom $C \sqsubseteq D \sqcap \exists r.E$ in these three syntaxes:

[2] The triple $\langle C, \mathit{rdfs}{:}\mathit{comment}, D\rangle$ is an example of an RDFS statement that has no correspondence with a TBox axiom; it states that D is a human-readable description of C.

Functional-style syntax

```
SubClassOf(
  :C
  ObjectIntersectionOf(
    :D
    ObjectSomeValuesFrom( :r :E))))
```

RDF/XML, an XML-based syntax for triples

```
<owl:Class rdf:about=":C">
  <rdfs:subClassOf>
    <owl:Class>
      <owl:intersectionOf rdf:parseType="Collection">
        <rdf:Description rdf:about=":D"/>
        <owl:Restriction>
          <owl:onProperty rdf:resource=":r"/>
          <owl:someValuesFrom rdf:resource=":E"/>
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
  </rdfs:subClassOf>
</owl:Class>
```

Manchester syntax

```
Class: :C
  SubClassOf: :D and (:r some :E)
```

The OWL specification also includes two different methods of defining the semantics of OWL ontologies. The *direct* semantics is defined with respect to the functional-style syntax, and hence is only applicable to RDF graphs that can be mapped into an OWL functional-style syntax ontology; such ontologies are referred to as OWL (2) DL ontologies. The *RDF-based* semantics is defined directly on RDF graphs, and is applicable to any graph, even those that include apparently malformed OWL syntax, or nonsensical triples such as $\langle rdf{:}type, rdf{:}type, rdf{:}type \rangle$; in the 2004 version of OWL, such ontologies are referred to as OWL Full, but in OWL 2 they are referred to as OWL 2 ontologies interpreted under the RDF-based semantics.

It is easy to show that all standard reasoning problems are, in general, undecidable for OWL Full ontologies (which can only be interpreted using the RDF-based semantics) [Mot07]. Perhaps for this reason, most

Axiom	Syntax	Semantics
Complex role		
Inclusion (CRIA)	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^{\mathcal{I}} \circ \ldots \circ R_n^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Disjointness	$\mathsf{Disj}(R, S)$	$R^{\mathcal{I}} \cap S^{\mathcal{I}} = \emptyset$
Transitivity	$\mathsf{Trans}(R)$	$R^{\mathcal{I}} \circ R^{\mathcal{I}} \subseteq R^{\mathcal{I}}$
Reflexivity	$\mathsf{Ref}(R)$	$\{(x, x) \mid x \in \Delta^{\mathcal{I}}\} \subseteq R^{\mathcal{I}}$
Irreflexivity	$\mathsf{Irref}(R)$	$\{(x, x) \mid x \in \Delta^{\mathcal{I}}\} \cap R^{\mathcal{I}} = \emptyset$
Symmetry	$\mathsf{Sym}(R)$	$(x, y) \in R^{\mathcal{I}} \Rightarrow (y, x) \in R^{\mathcal{I}}$
Antisymmetry	$\mathsf{Asym}(R)$	$(x, y) \in R^{\mathcal{I}} \Rightarrow (y, x) \notin R^{\mathcal{I}}$

Table 8.1. *$\mathcal{SROIQ}$ role axioms.*

OWL tools support only OWL DL interpreted under the direct semantics. In the remainder of this chapter we will only consider the OWL DL setting, and we will treat OWL as being synonymous with OWL 2 DL.

8.1.2 OWL and $\mathcal{SROIQ}$

As mentioned above, OWL 2 corresponds closely to the $\mathcal{SROIQ}$ description logic. Before describing the features of OWL, it will therefore be useful to briefly introduce $\mathcal{SROIQ}$. The $\mathcal{S}$ in $\mathcal{SROIQ}$ is a widely used abbreviation for $\mathcal{ALC}$ extended with transitive roles (see the Appendix), the letter $\mathcal{R}$ denotes an extended set of role axioms, sometimes called a role box (RBox), and $\mathcal{O}$, $\mathcal{I}$ and $\mathcal{Q}$ denote, respectively, nominals, inverse roles and qualified number restrictions as introduced in Chapter 2.

So far, we have considered DLs with a range of constructors for building concept descriptions, but with only two constructors for roles: inverse and transitive. Adding an RBox partly redresses the balance by providing a generalisation of role inclusion axioms (RIAs) called complex role inclusion axioms (CRIAs), as well as axioms asserting that roles are disjoint, transitive, reflexive, irreflexive, symmetric or antisymmetric. In addition, $\mathcal{SROIQ}$ provides concepts of the form $\exists R.\mathsf{Self}$, which can be used to express "local reflexivity" of a role r, negated role assertions, i.e., assertions of the form $(\mathsf{Mary}, \mathsf{Ph456}) : \neg \mathit{teaches}$, which states that Mary does *not* teach Ph456, and the universal (or top) role, denoted U.[3]

Definition 8.1 ($\mathcal{SROIQ}$ RBox)**.** Let $\mathbf{R}$ be a set of role names, with $U \in \mathbf{R}$. A $\mathcal{SROIQ}$ *role* R is either a role name or the inverse S^- of a

[3] In any interpretation $\mathcal{I}$, U is interpreted as $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

role name S. For R, R_i and S $\mathcal{SROIQ}$ roles, a $\mathcal{SROIQ}$ *role axiom* is an expression of one of the forms given in the second column of Table 8.1; a $\mathcal{SROIQ}$ *role box* is a set of such axioms. In any interpretation $\mathcal{I}$, the universal role U is interpreted as $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a $\mathcal{SROIQ}$ role axiom if it satisfies the condition given in the third column of Table 8.1, where $\circ$ denotes the composition of two relations; i.e., for $R^{\mathcal{I}}, S^{\mathcal{I}}$ binary relations, we define

$$R^{\mathcal{I}} \circ S^{\mathcal{I}} = \{(e, g) \mid \text{there is some } f \text{ with } (e, f) \in R^{\mathcal{I}} \text{ and } (f, g) \in S^{\mathcal{I}}\}.$$

An interpretation $\mathcal{I}$ *satisfies* a $\mathcal{SROIQ}$ RBox $\mathcal{R}$ if it satisfies each axiom in $\mathcal{R}$; such an interpretation is called a *model* of $\mathcal{R}$.

Please note that a CRIA as defined in Table 8.1 and with $n = 1$ is a role inclusion axiom (RIA) as introduced in Section 2.5.4.

Before we discuss further syntactic restrictions, let us consider an example RBox which captures some axioms concerning family relationships and partonomic ones:

$$\begin{aligned}
\mathit{hasMother} &\sqsubseteq \mathit{hasParent},\\
\mathit{hasSon} &\sqsubseteq \mathit{hasChild},\\
\mathit{hasChild} &\sqsubseteq \mathit{childOf}^{-},\\
\mathit{childOf} &\sqsubseteq \mathit{hasChild}^{-},\\
\mathit{hasParent} \circ \mathit{hasBrother} &\sqsubseteq \mathit{hasUncle},\\
\mathit{hasParent} &\sqsubseteq \mathit{hasAncestor},\\
\mathsf{Trans}(\mathit{hasAncestor}),&\\
\mathsf{Disj}(\mathit{hasSibling}, \mathit{childOf}),&\\
\mathsf{Irref}(\mathit{childOf}),&\\
\mathsf{Asym}(\mathit{childOf}),&\\
\mathit{isLocatedIn} \circ \mathit{isPartOf} &\sqsubseteq \mathit{isLocatedIn},\\
\mathsf{Trans}(\mathit{isPartOf}).&
\end{aligned}$$

While the axioms regarding family relations should be self-explanatory, it is worth pointing out the effect of the last two axioms, which motivated the support of complex inclusions in DLs and OWL [HS04]. For example, the last two axioms above together with the following concept inclusions:

$$\begin{aligned}
\mathsf{FracOfFemur} &\equiv \mathsf{Fracture} \sqcap \exists \mathit{isLocatedIn}.\mathsf{Femur},\\
\mathsf{FracOfHeadOfFemur} &\equiv \mathsf{Fracture} \sqcap \exists \mathit{isLocatedIn}.\mathsf{HeadOfFemur},\\
\mathsf{HeadOfFemur} &\sqsubseteq \mathsf{BodyPart} \sqcap \exists \mathit{isPartOf}.\mathsf{Femur},\\
\mathsf{Femur} &\sqsubseteq \mathsf{BodyPart} \sqcap \exists \mathit{isPartOf}.\mathsf{Leg},
\end{aligned}$$

entail

$$\begin{aligned} \mathsf{HeadOfFemur} &\sqsubseteq \exists \mathit{isPartOf}.\mathsf{Leg}, \\ \mathsf{FracOfHeadOfFemur} &\sqsubseteq \mathsf{FracOfFemur}. \end{aligned}$$

Further to Definition 8.1, to ensure that reasoning over $\mathcal{SROIQ}$ is decidable, $\mathcal{SROIQ}$ restricts RBoxes to *regular* ones and defines what it means for a role to be *simple* [HKS06]. Both conditions are rather tedious and technical, so we will only give an informal description here. For regularity, we know that the unrestricted use of CRIAs already leads to undecidability in $\mathcal{SHIQ}$ [HS04]. The regularity condition[4] ensures that the interactions between role names as enforced by an RBox can be captured by finite state automata which can then be used in a tableau algorithm. For simple roles, please note that role axioms such as $\mathsf{Trans}(S)$ or $R_1 \circ \cdots \circ R_n \sqsubseteq S$ imply "shortcuts"; for example, in any model $\mathcal{I}$ of $\mathsf{Trans}(S)$, an $S^{\mathcal{I}}$ path $\{(e_0, e_1), (e_1, e_2), \ldots, (e_{n-1}, e_n)\} \subseteq S^{\mathcal{I}}$ from e_0 to e_n implies a shortcut $(e_0, e_n) \in S^{\mathcal{I}}$. Using roles such as S for which shortcuts are implied in number restrictions is another source of undecidability [HST99]. To restore decidability, *simple* roles are defined as those for which no shortcuts are implied (e.g., that do not occur in transitivity axioms, and whose inverses also do not occur in transitivity axioms), and only simple roles can be used in role irreflexivity, antisymmetry and disjointness axioms, and in certain concept descriptions, as specified in the following definition.

Definition 8.2 ($\mathcal{SROIQ}$ Concepts)**.** Let $\mathbf{C}$ and $\mathbf{I}$ be disjoint sets of, respectively, *concept names* and *individual names*, with both $\mathbf{C}$ and $\mathbf{I}$ disjoint from $\mathbf{R}$. The set of $\mathcal{SROIQ}$ *concept descriptions* over $\mathbf{C}$ and $\mathbf{I}$ is inductively defined as follows:

- every concept name is a $\mathcal{SROIQ}$ concept description;
- $\top$ and $\bot$ are $\mathcal{SROIQ}$ concept descriptions;
- if C and D are $\mathcal{SROIQ}$ concept descriptions, R is a $\mathcal{SROIQ}$ role, S is a simple $\mathcal{SROIQ}$ role and n is a non-negative number, then the following are also $\mathcal{SROIQ}$ concept descriptions:
  - $C \sqcap D$ (conjunction),
  - $C \sqcup D$ (disjunction),
  - $\neg C$ (negation),
  - $\exists R.C$ (existential restriction),
  - $\forall R.C$ (value restriction),

[4] Interestingly, it has recently been shown that the version of these conditions given in the OWL 2 standard [OWL12c] is insufficient [Ste15].

- $\exists S.\mathsf{Self}$ (self restriction),
- $(\geqslant n\, S.C)$ (qualified number restriction), and
- $(\leqslant n\, S.C)$ (qualified number restriction).

Given an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, the mapping $\cdot^{\mathcal{I}}$ is extended to self restrictions as follows:

$$(\exists S.\mathsf{Self})^{\mathcal{I}} := \{d \in \Delta^{\mathcal{I}} \mid (d,d) \in S^{\mathcal{I}}\}.$$

Other concept descriptions are interpreted as per the definitions given in Chapter 2.

The only new concept constructor in Definition 8.2 is the self restriction: we can use it, for example, to describe people who love themselves by $\mathsf{Person} \sqcap \exists \mathit{loves}.\mathsf{Self}$.

Next, we define a $\mathcal{SROIQ}$ knowledge base: in addition to a TBox and an ABox, it also contains an RBox; the notion of "satisfaction" and "model" are extended to these in the usual way. It is a matter of taste whether we prefer to have three separate boxes or to allow role axioms in the TBox: here, we have opted for the former, but this choice is immaterial.

Definition 8.3 ($\mathcal{SROIQ}$ Knowledge Base)**.** For C and D $\mathcal{SROIQ}$ concept descriptions, $C \sqsubseteq D$ is a $\mathcal{SROIQ}$ *general concept inclusion* (GCI); a $\mathcal{SROIQ}$ *TBox* is a finite set of $\mathcal{SROIQ}$ GCIs. An interpretation $\mathcal{I}$ *satisfies* a $\mathcal{SROIQ}$ GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it *satisfies* a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ if it satisfies each GCI in $\mathcal{T}$; such an interpretation is called a *model* of $\mathcal{T}$.

For $a, b \in \mathbf{I}$ individual names, C a $\mathcal{SROIQ}$ concept description, and R a $\mathcal{SROIQ}$ role, $a : C$ is a $\mathcal{SROIQ}$ *concept assertion* and $(a,b) : R$ and $(a,b) : \neg R$ are $\mathcal{SROIQ}$ *role assertions*; a $\mathcal{SROIQ}$ *ABox* is a finite set of $\mathcal{SROIQ}$ concept and role assertions. An interpretation $\mathcal{I}$ *satisfies* $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, it *satisfies* $(a,b) : R$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, and it *satisfies* $(a,b) : \neg R$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \notin R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ *satisfies* a $\mathcal{SROIQ}$ ABox $\mathcal{A}$ if it satisfies each concept and role assertion in $\mathcal{A}$; such an interpretation is called a *model* of $\mathcal{A}$.

A $\mathcal{SROIQ}$ *knowledge base* $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ consists of a regular $\mathcal{SROIQ}$ RBox $\mathcal{R}$, TBox $\mathcal{T}$ and ABox $\mathcal{A}$; an interpretation $\mathcal{I}$ is a *model* of $\mathcal{K}$ if it is a model of each of $\mathcal{R}$, $\mathcal{T}$ and $\mathcal{A}$.

Note that several of the axioms described in Table 8.1 are redundant

in the sense that they could be expressed using other means, as captured by the following lemma.[5]

Lemma 8.4. *Let R, S be possibly inverse roles. Then we have the following:*

(i) $\mathsf{Trans}(R)$ *is equivalent to the CRIA* $R \circ R \sqsubseteq R$;
(ii) $\mathsf{Sym}(R)$ *is equivalent to* $R \sqsubseteq R^-$;
(iii) $\mathsf{Ref}(R)$ *is equivalent to* $\top \sqsubseteq \exists R.\mathsf{Self}$;
(iv) $\mathsf{Irref}(R)$ *is equivalent to* $\top \sqsubseteq \neg\exists R.\mathsf{Self}$.

8.1.3 OWL ontologies

An OWL ontology can be seen to correspond to a DL knowledge base $\mathcal{K} = (\mathcal{R}, \mathcal{T}, \mathcal{A})$ with its three boxes combined in a single set $\mathcal{R} \cup \mathcal{T} \cup \mathcal{A}$.[6] It is, however, trivial to sort this set into an RBox, TBox and ABox, and we will sometimes talk about an OWL TBox and ABox. In fact in the literature an OWL ontology is often assumed to be a TBox and an RBox, with the ABox assertions (if any) being stored separately as RDF triples.

As in a standard DL, an OWL TBox describes the domain in terms of *classes* (corresponding to concepts), *properties* (corresponding to roles) and *individuals* (corresponding to individual names), and consists of a set of *axioms* that assert, e.g., subsumption relationships between classes or properties.

As usual, OWL classes and properties may be names or expressions built up from simpler classes and properties using a variety of constructors. The main constructors supported by OWL, along with the equivalent DL syntax, are summarised in Table 8.2, where C (possibly subscripted) is a class, p is a property, x (possibly subscripted) is an individual and n is a non-negative integer. Note that:

- OWL provides an explicit bottom property (i.e., a property whose extension is empty in every interpretation), and an exact cardinality class constructor, but the semantics of these can be trivially simulated in $\mathcal{SROIQ}$. For example, if $\top \sqsubseteq \forall B.\bot \in \mathcal{T}$, then $B^{\mathcal{I}} = \emptyset$ in any model $\mathcal{I}$ of $\mathcal{T}$, and $(=n\, p.C)$ is equivalent to $(\geqslant n\, p.C) \sqcap (\leqslant n\, p.C)$.

[5] We remind the reader that two axioms α, β are equivalent if an interpretation satisfies α if and only if it satisfies β.
[6] The OWL specification uses "axiom" as a generic term for both TBox axioms and ABox assertions.

OWL functional syntax	DL syntax
`ObjectInverseOf`(p)	p^-
`ObjectPropertyChain`($p_1, \ldots, p_n$)	$p_1 \circ \ldots \circ p_n$
`owl:topObjectProperty`	U
`owl:bottomObjectProperty`	B
`owl:Thing`	$\top$
`owl:Nothing`	$\bot$
`ObjectIntersectionOf`($C_1 \ \ldots \ C_n$)	$C_1 \sqcap \ldots \sqcap C_n$
`ObjectUnionOf`($C_1 \ \ldots \ C_n$)	$C_1 \sqcup \ldots \sqcup C_n$
`ObjectComplementOf`(C)	$\neg C$
`ObjectOneOf`($x_1 \ \ldots \ x_n$)	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$
`ObjectAllValuesFrom`($p \ C$)	$\forall p.C$
`ObjectSomeValuesFrom`($p \ C$)	$\exists p.C$
`ObjectHasValue`($p \ x$)	$\exists p.\{x\}$
`ObjectHasSelf`(p)	$\exists p.Self$
`ObjectMinCardinality`($n \ p \ C$)	$(\geqslant n \, p.C)$
`ObjectMaxCardinality`($n \ p \ C$)	$(\leqslant n \, p.C)$
`ObjectExactCardinality`($n \ p \ C$)	$(=n \, p.C)$

Table 8.2. *OWL property and class constructors.*

- The use of the `ObjectPropertyChain` constructor is restricted to CRIAs of the form $p_1 \circ \ldots \circ p_n \sqsubseteq p$. In all other cases, OWL properties must be either property names (IRIs) or inverse properties.

An important feature of OWL is that, in addition to classes and individuals, the ontology can also use *datatypes* and *literals* (i.e., data values). The set of datatypes supported by OWL, including their syntax and semantics, is defined in the OWL 2 Datatype Map; most of these are taken from the set of XML Schema Datatypes (XSD) [XSD12], and include various number types (such as `xsd:float` and `xsd:integer`), string types (such as `xsd:string`), Booleans (`xsd:boolean`), IRIs (`xsd:anyURI`) and time instants (`xsd:dateTime` and `xsd:dateTimeStamp`). Literals may be either typed (e.g., `"42"^^xsd:integer`) or untyped (e.g., `"lifetheuniverseandeverything"`). OWL's datatypes and literals come with some useful "syntactic sugar", but semantically they can be seen as a restricted form of *concrete domains* which allows only unary predicates and feature paths of length one (see Section 5.3.2) [BH91, LAHS04, HS01].

Like classes, datatypes can be combined and constrained to form user-defined datatypes called *data ranges*. Each datatype is a data range, and many datatypes can additionally be constrained using *facets* such

as *xsd*:*minInclusive*; for example,

xsd:*integer* *xsd*:*minExclusive* `"15"^^`*xsd*:*integer*

defines the data range based on integer whose values include all those integers greater than 15. Data ranges can also be combined using Boolean constructors similar to those used with classes, i.e., DataIntersectionOf, DataUnionOf, DataComplementOf and DataOneOf; for example,

`DataUnionOf(` *xsd*:*string* *xsd*:*integer*)

specifies a data range that contains all strings and all integers. Finally, OWL *datatype definitions* provide a simple mechanism for naming data ranges; for example,

`DatatypeDefinition(` :*over15*
xsd:*integer* *xsd*:*minExclusive* `"15"^^`*xsd*:*integer*)

introduces the name :*over15* as an abbreviation for the data range consisting of integers greater than 15. Datatype definitions are restricted (e.g., to be acyclic) such that the datatypes they define can be treated as macros; i.e., given an ontology $\mathcal{O}$ containing the above datatype definition, other occurrences of :*over15* can be replaced with *xsd*:*integer* *xsd*:*minExclusive* `"15"^^`*xsd*:*integer* without affecting the semantics of $\mathcal{O}$.

As in DL Datatypes [HS01], OWL imposes a strict separation between classes and datatypes: the interpretation domain of classes is disjoint from that of datatypes, and the set of properties that relate pairs of individuals (called *object properties*) is disjoint from the set of properties that relate individuals to literals (called *data properties*). This ensures that reasoning algorithms can be relatively straightforwardly extended to support datatypes by employing a *datatype oracle* that decides basic reasoning problems about datatypes and literals [MH08]. Moreover, in order to avoid any syntactic ambiguity, OWL distinguishes class constructors used with classes and individuals ("object" constructors) from those used with datatypes and literals ("data" constructors); this allows object and data properties to be correctly typed without the need for typing declarations (OWL does allow for such declarations, but they are not mandatory); by dint of its occurrence in object constructors, the property p used in Table 8.2 is thus unambiguously an object property. Class constructors using data ranges and literals are otherwise similar to object constructors, and are shown in Table 8.3, where D (possibly

OWL functional syntax	DL syntax
`DataAllValuesFrom`(d D)	$\forall d.D$
`DataSomeValuesFrom`(d D)	$\exists d.D$
`DataHasValue`(d v)	$\exists d.\{v\}$
`DataMinCardinality`(n d D)	$(\geqslant n\, d.D)$
`DataMaxCardinality`(n d D)	$(\leqslant n\, d.D)$
`DataExactCardinality`(n d C)	$(=n\, d.D)$

Table 8.3. *OWL data property class constructors.*

subscripted) is a datatype, d is a data property, v is a data value and n is a non-negative integer.

The distinction between object and data properties is maintained in property axioms and assertions; e.g., there are distinct axioms for asserting subsumption between object properties and data properties. The axioms and assertions provided by OWL, along with the equivalent DL syntax, are summarised in Tables 8.4 and 8.5, where C (possibly subscripted) is a class, p (possibly subscripted) is an object property, d (possibly subscripted) is a data property, a (possibly subscripted) is an individual and v is a data value. Recall that in `SubObjectPropertyOf` axioms p_1 can be a property name (an IRI), an inverse property or a property chain; in all other cases properties are restricted to being property names or inverse properties.

Note that some OWL axioms are equivalent to sets of $\mathcal{SROIQ}$ axioms; for example, an OWL `EquivalentObjectProperties` axiom takes two or more object properties, and is semantically equivalent to two or more role inclusion axioms in $\mathcal{SROIQ}$. Similarly, we can say that two or more (object or data) properties are pairwise disjoint, e.g., in `DisjointObjectProperties`$(p_1 \ldots p_n)$; the $\mathcal{SROIQ}$ equivalent is a set of axioms of the form $p_i \sqsubseteq \neg p_j$, each of which states that a pair of roles are disjoint. The semantics of such axioms is straightforward: an interpretation $\mathcal{I}$ satisfies $p_i \sqsubseteq \neg p_j$ if $p_i^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus p_j^{\mathcal{I}}$ or, in other words, if $p_i^{\mathcal{I}} \cap p_j^{\mathcal{I}} = \emptyset$.

A final point to mention is that we can, explicitly, state that two or more individuals are the same via `SameIndividual`$(a_1 \ldots a_n)$ or that they are pairwise different via `DifferentIndividuals`$(a_1 \ldots a_n)$.

Axiom	DL Syntax
`SubObjectPropertyOf`($p_1\ p_2$)	$p_1 \sqsubseteq p_2$
`EquivalentObjectProperties`($p_1 \ldots p_n$)	$\cup_{i \neq j}\{p_i \sqsubseteq p_j\}$
`DisjointObjectProperties`($p_1 \ldots p_n$)	$\cup_{i \neq j}\{p_i \sqsubseteq \neg p_j\}$
`InverseObjectProperties`($p_1\ p_2$)	$p_1 \equiv p_2^-$
`ObjectPropertyDomain`($p\ C$)	$\exists p.\top \sqsubseteq C$
`ObjectPropertyRange`($p\ C$)	$\top \sqsubseteq \forall p.C$
`FunctionalObjectProperty`(p)	$\top \sqsubseteq (\leqslant 1\, p)$
`InverseFunctionalObjectProperty`(p)	$\top \sqsubseteq (\leqslant 1\, p^-)$
`ReflexiveObjectProperty`(p)	$\mathsf{Ref}(p)$
`IrreflexiveObjectProperty`(p)	$\mathsf{Irref}(p)$
`SymmetricObjectProperty`(p)	$\mathsf{Sym}(p)$
`AsymmetricObjectProperty`(p)	$\mathsf{Asym}(p)$
`TransitiveObjectProperty`(p)	$\mathsf{Trans}(p)$
`SubDataPropertyOf`($d_1\ d_2$)	$d_1 \sqsubseteq d_2$
`EquivalentDataProperties`($d_1 \ldots d_n$)	$\cup_{i \neq j}\{d_i \sqsubseteq d_j\}$
`DisjointDataProperties`($d_1 \ldots d_n$)	$\cup_{i \neq j}\{d_i \sqsubseteq \neg d_j\}$
`DataPropertyDomain`($d\ C$)	$(\geqslant 1\, d) \sqsubseteq C$
`DataPropertyRange`($d\ D$)	$\top \sqsubseteq \forall d.D$
`FunctionalDataProperty`(d)	$\top \sqsubseteq (\leqslant 1\, d)$

Table 8.4. *OWL property axioms, where unions range over i, j between 1 and n.*

Axiom	DL Syntax
`SubClassOf`($C_1\ C_2$)	$C_1 \sqsubseteq C_2$
`EquivalentClasses`($C_1 \ldots C_n$)	$\cup_{i \neq j}\{C_i \sqsubseteq C_j\}$
`DisjointClasses`($C_1 \ldots C_n$)	$\cup_{i \neq j}\{C_i \sqsubseteq \neg C_j\}$
`DisjointUnion`($C\ C_1 \ldots C_n$)	$\cup_{i \neq j}\{C_i \sqsubseteq \neg C_j\} \cup \{C \equiv C_1 \sqcup \ldots \sqcup C_n\}$
`SameIndividual`($a_1 \ldots a_n$)	$\cup_{i \neq j}\{a_i = a_j\}$
`DifferentIndividuals`($a_1 \ldots a_n$)	$\cup_{i \neq j}\{a_i \neq a_j\}$
`ClassAssertion`($C\ a$)	$a : C$
`ObjectPropertyAssertion`($p\ a_1\ a_2$)	$(a_1, a_2) : p$
`NegativeObjectPropertyAssertion`($p\ a_1\ a_2$)	$(a_1, a_2) : \neg p$
`DataPropertyAssertion`($d\ a\ v$)	$(a, v) : d$
`NegativeDataPropertyAssertion`($d\ a\ v$)	$(a, v) : \neg d$

Table 8.5. *OWL class axioms and assertions, where unions range over i, j between 1 and n.*

8.1.4 Non-DL features

Although largely a syntactic variant of $\mathcal{SROIQ}$, OWL also includes a number of features that are not found in standard DLs.

Keys

OWL ontologies can additionally include `HasKey` axioms, the purpose of which is to provide funcionality similar to keys in relational databases. A `HasKey` axiom is of the form

$$\texttt{HasKey}(\ C\ (\ p_1\ \ldots p_n\ )\ (\ d_1\ \ldots d_m)\ ),$$

where C is a class, p_i is an object property and d_j is a data property. Such an axiom states that no two distinct *named* instances of class C can be related to the same set of individuals and literals via the given properties, i.e., that *named* instances of C are uniquely identified by these relationships, where an individual is *named* if it occurs syntactically in the ontology.

More precisely, given an ontology $\mathcal{O}$ with a `HasKey` axiom

$$\texttt{HasKey}(\ C\ (\ p_1\ \ldots p_n\ )\ (\ d_1\ \ldots d_m)\ ) \in \mathcal{O},$$

a model $\mathcal{I}$ of $\mathcal{O}$ has to satisfy the following condition: if a, b are individuals occurring in $\mathcal{O}$, $\{a^{\mathcal{I}}, b^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$, and for each $e \in \Delta^{\mathcal{I}}$, $v \in \Delta^{\mathsf{D}}$, $i \leq n$, and $j \leq m$, we have

- $(a^{\mathcal{I}}, e) \in p_i^{\mathcal{I}}$ if and only if $(b^{\mathcal{I}}, e) \in p_i^{\mathcal{I}}$, and
- $(a^{\mathcal{I}}, v) \in d_j^{\mathcal{I}}$ if and only if $(b^{\mathcal{I}}, v) \in d_j^{\mathcal{I}}$,

then $a^{\mathcal{I}} = b^{\mathcal{I}}$.

For example, if an ontology $\mathcal{O}$ includes the following axiom and assertions:

`HasKey`(:*Person* (:*hasChild*) (:*hasGender*)),
`ClassAssertion`(:*Person* :*Elizabeth*),
`ObjectPropertyAssertion`(:*hasChild* :*Elizabeth* :*Mary*),
`DataPropertyAssertion`(:*hasGender* :*Elizabeth* `"F"`),
`ClassAssertion`(:*Person* :*Liz*),
`ObjectPropertyAssertion`(:*hasChild* :*Liz* :*Mary*),
`DataPropertyAssertion`(:*hasGender* :*Liz* `"F"`),

then $\mathcal{O}$ entails `SameIndividual`(:*Elizabeth* :*Liz*). If $\mathcal{O}$ additionally includes the following axioms and assertions:

`ClassAssertion`(`ObjectSomeValuesFrom`(*hasFriend* :*P*) :*John*),
`SubClassOf`(:*P* `ObjectHasValue`(*hasChild* :*Mary*)),
`SubClassOf`(:*P* `DataHasValue`(*hasGender* `"F"`)),
`SubClassOf`(:*P* :*Person*), `SubClassOf`(:*P* :*Happy*),
`ClassAssertion`(`ObjectComplementOf`(:*Happy*) :*Liz*),

then it might at first appear that Peter has at least one friend who is also entailed to be the same individual as :*Elizabeth* and :*Liz* (because they too have :*Mary* as their child and `"F"` as their gender), and that when combined with the fact that Peter's friend is :*Happy* while :*Liz* is ¬:*Happy*, this would make $\mathcal{O}$ inconsistent. However, the key axiom does not apply to Peter's friend, because this friend is not explicitly named in $\mathcal{O}$, and so does not lead to an inconsistency.

Anonymous individuals

As we saw in Section 8.1.1, ABox assertions in OWL directly correspond to RDF triples of the form $\langle a, rdf{:}type, C\rangle$ and $\langle a, p, b\rangle$, where C is a class, p is a property and a and b are IRIs. Unlike standard DLs, a and b do not have to be named individuals, but can also be RDF *blank nodes*. Blank nodes are denoted by the use of _: as an IRI prefix (e.g., _:x), and are treated as variables that are existentially quantified at the outer level of the ABox [MAHP11]. In OWL, blank nodes used in ABox assertions are called *anonymous individuals*. For example, the assertions

`ObjectPropertyAssertion`(:*hasFriend* :*Liz* _:x),
`ObjectPropertyAssertion`(:*livesIn* _:x _:y),
`ObjectPropertyAssertion`(:*livesIn* :*Mary* _:y)

assert that :*Liz* has a friend who lives in the same place as :*Mary* without explicitly naming the friend or the place where they live; they are semantically equivalent to a first-order logic sentence of the form

$$\exists x \exists y\, (hasFriend(Liz, x) \wedge livesIn(x, y) \wedge livesIn(Mary, y)).$$

These assertions can also be written as the semantically equivalent $\mathcal{SROIQ}$ concept assertion

$$Liz : \exists hasFriend.(\exists livesIn.(\exists livesIn^{-}.\{Mary\})),$$

and hence can be similarly written in OWL without recourse to blank nodes.

This rewriting procedure, where existential restrictions are used to transform property assertions into semantically equivalent class assertions, is often referred to in the literature as *rolling up* [HT00]. Rolling up can be used to eliminate anonymous individuals, as in the above example, only if the property assertions that connect them have a tree-like structure, i.e., provided that anonymous individuals are not cyclically

connected. For example, if we extended the above ABox with the assertion

$$\texttt{ObjectPropertyAssertion}(\ :bornIn\ \_{:}x\ \_{:}y\ ),$$

then it would no longer be possible to use rolling up to eliminate $\_{:}x$ and $\_{:}y$.

OWL 2 DL ontologies must satisfy syntactic restrictions on the use of anonymous individuals which ensure that rolling up is always possible; hence any OWL 2 DL ontology $\mathcal{O}$ can be rewritten as a semantically equivalent OWL 2 DL ontology $\mathcal{O}'$ in which there are no anonymous individuals.

Metamodelling

In some applications it may be desirable to use the same name for both a class (or property) and an individual. For example, we might want to state that :*Harry* is an instance of :*Eagle*

$$\texttt{ClassAssertion}(\ :Eagle\ :Harry\ )$$

and that :*Eagle* is an instance of :*EndangeredSpecies*

$$\texttt{ClassAssertion}(\ :EndangeredSpecies\ :Eagle\ ).$$

We could then extend our modelling of the domain to describe classes of classes, e.g., by stating that it is illegal to hunt any class of animal that is an instance of :*EndangeredSpecies*; this is often called *metamodelling*. Metamodelling is not possible in a standard DL, where it is usually assumed that the sets $\mathbf{C}$, $\mathbf{R}$ and $\mathbf{I}$ (of, respectively, concept, role and individual names) are pairwise disjoint, and where class assertions can only be used to describe individual names; i.e., in an assertion $a : C$, a must be an individual name.

OWL 2 uses a mechanism known as *punning* to provide a simple form of metamodelling while still retaining the correspondence between OWL ontologies and $\mathcal{SROIQ}$ KBs. Punning allows for the same IRI (name) to be used as an individual, a class and a property, but it applies the *contextual semantics* described by Motik in [Mot07]. In the contextual semantics, IRIs used in the individual, class and property contexts are semantically unrelated; this semantics is equivalent to rewriting the ontology by adding unique prefixes such as i:, c: and p: to IRIs according to the context in which they occur. For example, the above assertions would be treated as though they were written

$$\texttt{ClassAssertion}(\ c{:}Eagle\ i{:}Harry\ )$$

and

$$\texttt{ClassAssertion}(\ c{:}\mathit{EndangeredSpecies}\ i{:}\mathit{Eagle}\ ),$$

with $c{:}\mathit{Eagle}$ a class name and $i{:}\mathit{Eagle}$ an individual name. This is easy to achieve as the context of each IRI occurrence is clear from the syntactic structure of the ontology. Punning thus has no effect on standard reasoning tasks (such as classification), but it does allow for queries that, e.g., return individuals that are instances of species that are themselves instances of $c{:}\mathit{EndangeredSpecies}$.

Annotations

OWL includes a flexible annotation mechanism that allows for comments and other "non-logical" information to be included in the ontology. An OWL annotation consists of an annotation property and a literal, and zero or more annotations can be attached to class, property and individual names, to axioms and assertions, to datatypes, to the ontology as a whole and even to annotations themselves; for example,

$$\texttt{ClassAssertion}(\ \texttt{Annotation}(\ \mathit{rdfs{:}comment}\ \texttt{"Liz is a person"}\ )\ :\mathit{Person}\ :\mathit{Liz}\ )$$

annotates the class assertion with the property $\mathit{rdfs{:}comment}$ and the literal `"Liz is a person"`.

Annotation properties can be used to distinguish different kinds of annotations, with OWL even providing for a basic type structure via annotation property specific range, domain and sub-property axioms. Note, however, that annotations and annotation property axioms have no formal semantics, and can simply be discarded when translating the ontology into a DL knowledge base. As with other OWL properties, annotation property names are IRIs, and the set of annotation property names is pairwise disjoint from the sets of object property and data property names.

Imports

Each OWL ontology is associated with an *ontology document* in which the various statements that make up the ontology are stored. OWL makes no assumptions about the structure of such documents, but it is assumed that each ontology document can be accessed via an IRI, and that its contents can be converted into an ontology. For the sake of brevity, we will from now on refer to ontology documents using the IRIs

via which they are accessed; for example, we will refer to the ontology document that can be accessed via :*ont* simply as :*ont*.

The OWL `Import` statement provides a mechanism for "importing" the contents of one ontology document into another; for example, if :*ont1* includes the statement

`Import`(:*ont2*),

then :*ont1* is treated as though it also includes all of the contents of :*ont2* and, recursively, any ontology documents imported by :*ont2*. The OWL specification defines a parsing procedure that extracts ontological content from the current ontology document and all those that it (possibly recursively) imports, while ensuring termination even if ontology documents (directly or indirectly) import each other cyclically.

8.1.5 OWL profiles

An important change in OWL 2 was the introduction of *profiles*. A profile is "a trimmed down version of OWL 2 that trades some expressive power for efficiency of reasoning" [OWL12b], i.e., a syntactic subset (sometimes called a fragment) of the language that enjoys better computational properties. Three profiles are defined: OWL 2 EL, OWL 2 QL and OWL 2 RL, each of which provides different expressive power and targets different application scenarios. The OWL 2 profiles are defined by placing restrictions on the functional-style syntax of OWL 2; in OWL 2 EL, for example, one such restriction forbids the use of the `ObjectComplementOf` (class negation) constructor in class expressions.

Note that the original OWL language specification also defined a subset, called OWL Lite. The computational properties of this subset are, however, only marginally better than those of the unrestricted language (ontology satisfiability is ExpTime-complete [HPSvH03]); as a result OWL Lite was little used, and was not included as one of the OWL 2 profiles.

OWL 2 EL is based on $\mathcal{EL}^{++}$, a family of description logics that extend $\mathcal{EL}$ (see Chapter 6) while ensuring that satisfiability and subsumption with respect to general TBoxes remains polynomial in the size of the TBox [BBL05]. Optimised implementations of PTime algorithms for TBox classification have proved to be very effective in practice, and are widely used in the development of healthcare and life science ontologies, including the SNOMED healthcare ontology which is developed

and maintained by the International Health Terminology Standards Development Organisation [BLS06, KKS11, SSBB09].

OWL 2 QL is based on the DL-Lite family of description logics [ACKZ09], for which conjunctive queries are FO-rewritable, and for which conjunctive query answering is thus in AC^0 with respect to the size of the data. More specifically, OWL 2 QL is based on DL-Lite$_{\mathcal{R}}$, a variant of DL-Lite that additionally allows for role inclusion and role disjointness axioms. FO-rewritability allows for query answering to be implemented on top of relational database systems, with query evaluation being delegated to the DB system.

OWL 2 RL is based on *description logic programs* [GHVD03], a logic that aims to capture the intersection between Description Logic and Datalog, i.e., a description logic whose TBox axioms can be translated into Datalog rules. As the resulting language can be seen as a subset of Datalog, query answering is in PTIME with respect to the size of the data [DEGV01]; moreover, implementations can exploit existing rule engines, several of which have been shown to be highly scalable in practice [BKO+11, MNP+14].

8.2 OWL tools and applications

The correspondence between OWL and Description Logic means that DL algorithms and systems can be used to provide reasoning services for OWL tools and applications. A wide range of DL-based OWL reasoners is available, including both general-purpose and profile-specific systems (see, e.g., `http://www.w3.org/2001/sw/wiki/OWL/Implementations` and `http://owl.cs.manchester.ac.uk/tools/list-of-reasoners/` for lists maintained by, respectively, the W3C and the University of Manchester). On the other hand, OWL tools and infrastructure provide convenient and practical mechanisms for both developing and deploying DL knowledge bases. In the following we will briefly mention a few prominent and interesting examples of OWL tools and applications.

8.2.1 The OWL API

The OWL API is a Java API and reference implementation for creating, manipulating and serialising OWL Ontologies (see `http://owlcs.github.io/owlapi/`). Although not a tool or application per se, the

OWL API is an important component of numerous tools and applications, and is widely used for parsing and writing OWL ontologies in various syntaxes (including RDF/XML), and for interfacing with reasoners.

8.2.2 OWL reasoners

As mentioned above, a wide range of DL-based OWL reasoners is available, including both general-purpose and profile-specific systems. Currently, all fully fledged OWL reasoners (i.e., those that support most or all of the OWL language), are based on tableau algorithms similar to those described in Chapter 4, although efforts are being made to extend the consequence-based techniques described in Chapter 6 to larger fragments of OWL [SKH11, BMG+15]. Prominent examples of tableau-based OWL reasoners include FaCT++ [TH06], HermiT [GHM+14], Konclude [SLG14] and Pellet [SPC+07].

Several profile-specific reasoners are also available. For the OWL 2 EL profile, most reasoners are based on consequence-based techniques as described in Chapter 6; prominent examples include CEL [BLS06], ELK [KKS14] and SnoRocket [MJL13]. However, there are also several systems for query answering over RDF data with respect to (subsets of) OWL 2 EL ontologies that use rewriting techniques similar to those described in Section 7.3; these include REQUIEM [PUMH10], KARMA [SMH13] and EOLO [SM15]. For the OWL 2 QL profile, most systems are based on the query rewriting techniques described in Chapter 7; prominent examples include Mastro [CCD+13], Grind [HLSW15] and Ontop [KRR+14]. Such systems typically answer (unions of) conjunctive queries with respect to an OWL 2 QL ontology and data stored in a relational database. For the OWL 2 RL profile, most systems exploit Datalog reasoning techniques, including both forward chaining (also known as materialisation) and backwards chaining; prominent examples include GraphDB [BKO+11] (formerly known as OWLIM), RDFox [MNP+14] and Oracle's RDF store [WED+08]. Such systems typically answer SPARQL queries [SPA13] with respect to an OWL 2 RL ontology, where the data may be stored separately as RDF triples.

8.2.3 Ontology engineering tools

Numerous tools are available for developing and maintaining OWL ontologies (see `http://www.w3.org/wiki/Ontology_editors`). Prominent examples include Protégé [KFNM04], a "free, open-source ontology

editor and framework" developed by the Center for Biomedical Informatics Research at Stanford University School of Medicine, and TopBraid Composer, a commercial ontology "modelling environment" developed by TopQuadrant (see `http://www.topquadrant.com//`).

Protégé has played an important role in the popularisation of OWL by providing a sophisticated ontology development environment that is freely available for download (see `http://protege.stanford.edu/`). Protégé uses reasoning to support the development and maintenance process, e.g., checking for inconsistent classes, discovering implicit subsumption relationships and answering queries over the ABox. Protégé interfaces to reasoners via the OWL API, and so can exploit a wide range of reasoners, including many of those mentioned above.

Tools are also available for managing various aspects of ontology evolution, including ontology versioning [JRCHB11], merging [JRCZH12] and modularisation [JGS+08].

8.2.4 OWL applications

The availability of tools and systems, including those mentioned above, has contributed to the increasingly widespread use of OWL, and it is currently by far the most widely used ontology language, with applications in fields as diverse as agriculture [SLL+04], astronomy [DeRP06], biology [RB11, OSRM+12], defence [LAF+05], education [CBV+14], energy management [CGH+13], geography [Goo05], geoscience [RP05], medicine [CSG05, GZB06, HDG12, TNNM13], oceanography [KHJ+15b] and oil and gas [SLH13, KHJ+15a]. We discuss below a few representative applications, but this is very far from an exhaustive survey; interested readers should investigate the "industry" and/or "applications" tracks that are often organised by semantic web conferences (e.g., the International Semantic Web Conference[7] and European Semantic Web Conference[8]), and specialised conferences and journals in relevant areas (e.g., the *Journal of Biomedical Semantics*[9]).

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 Biological Pathways Exchange (BioPAX) ontology [RRL05], the GALEN ontology [RR06], the Foundational Model of Anatomy (FMA) [GZB06] and the National Cancer Institute thesaurus [HdD+05]. The National Centre for Biomedical Ontol-

[7] `http://swsa.semanticweb.org/`
[8] `http://www.eswc-conferences.org`
[9] `https://jbiomedsem.biomedcentral.com/`

ogy (see http://www.bioontology.org/) supports the ongoing development and maintenance of Protégé, and provides numerous resources, including a repository of biomedical ontologies (called BioPortal) and ontology-based tools for accessing and analysing biomedical data. The BioPortal repository contains several hundred ontologies, almost all of which are available in OWL and/or OBO formats, the latter being a text-based ontology language developed in the Open Biomedical Ontologies project and corresponding to a subset of OWL [GHH+07].

The SNOMED CT ontology is particularly noteworthy as it is very large (more than 300,000 classes) and is used in the healthcare systems of many countries (see http://www.ihtsdo.org/snomed-ct). The ontology is developed and maintained by the International Health Terminology Standards Development Organisation (IHTSDO), which is funded by member organisations from (at the time of writing) 27 countries. SNOMED CT uses a bespoke syntax, but this can be directly translated into OWL 2 EL, and reasoners such as ELK and SnoRocket are used to support the development and adaptation of SNOMED CT.

The importance of reasoning support in biomedical applications was highlighted in [KFP+06], which describes 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. This check revealed "systematic modelling errors", and a significant number of missed subClass 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".

Another important application of OWL is in tools that help non-expert users to access data stored in relational databases, a technique that is often called ontology-based data access (OBDA). In the EU Optique project (see http://optique-project.eu/), for example, OBDA was used to help geologists and geophysicists at the Norwegian oil and gas company Statoil to access data gathered from past and present operations and stored in large and complex relational databases; their Exploration and Production Data Store (EPDS), for example, stores around 700GB of data in more than 3,000 tables [KHJ+15a]. In the Optique system, an OWL 2 QL ontology provides a more user-friendly schema for query formulation, and the Ontop query rewriting system is then used to answer these queries over the EPDS database.

The Électricité de France (EDF) Energy Management Adviser (EMA)

uses the HermiT OWL reasoner to produce personalised energy saving advice for EDF's customers. The EMA uses an OWL ontology to model both relevant features of the domain (housing, environment, and so on) and a range of energy-saving "tips". Customers are then described using RDF, and SPARQL queries are used to generate a personalised set of tips for each customer. The system has been used to provide tips to more than 300,000 EDF customers in France.