Available online at www.sciencedirect.com
SCIENCE DIRECT®
Web Semantics: Science, Services and Agents on the World Wide Web 4 (2006) 29–39
JOURNAL OF Web Semantics
www.elsevier.com/locate/websem

OWL-Eu: Adding customised datatypes into OWL ☆

Jeff Z. Pan*, Ian Horrocks

School of Computer Science, University of Manchester, Kilburn Building, Manchester M13 9PL, UK

Received 9 May 2005; received in revised form 20 July 2005; accepted 10 August 2005

Abstract

Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome, and the W3C Semantic Web Best Practices and Development Working Group has set up a task force to address this issue. This paper makes the following two contributions: (i) it provides a brief summary of OWL-related datatype formalisms, and (ii) it provides a decidable extension of OWL DL, called OWL-Eu, that supports customised datatypes. A detailed proof of the decidability of OWL-Eu is presented.

Keywords: Ontologies; Semantic Web; Description Logics; Customised Datatypes; Unary datatype groups

1. Introduction

The OWL Web Ontology Language [3] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support [17,18] is one of the key features that OWL is expected to provide, and has prompted extensive discussions in the RDF-Logic mailing list [21] and in the Semantic Web Best Practices mailing list [23]. Although OWL adds considerable expressive power to the Semantic Web, the OWL datatype formalism (or simply *OWL datatyping*) is much too weak for many applications; in particular, OWL datatyping does not provide a general framework for customised datatypes,[1] such as XML Schema derived datatypes.

It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome [22], as it is often necessary to enable users to define their own datatypes and datatype predicates for their ontologies and applications.

Example 1. Customised datatypes are important in capturing the intended meaning of some vocabulary in ontologies. For example, the customised datatype 'atLeast18' can be used in the following definition of the class 'Adult':

```
Class(Adult complete Person
  restriction(age allvalueFrom
    (atLeast18))),
```

which says that an Adult is a Person whose *age* is at least 18. The datatype constraint 'at least 18' can be defined as an XML Schema user-defined datatype as follows:

```
<simpleType name="atLeast18">
  <restriction base=''xsd:integer">
    <minInclusive value=''18"/>
  </restriction>
</simpleType>
```
[2]

Such user-defined datatypes cannot, however, be used in OWL.

After reviewing the design of OWL, and the needs of various applications and (potential) users, the following requirements for an extension to OWL DL have been identified:

1. It should provide customised datatypes; therefore, it should be based on a datatype formalism which is compatible with OWL datatyping, provides facilities to construct customised

☆ This is a revised and extended version of a paper with the same title that was published in the Second European Semantic Web Conference (ESWC2005).

* Corresponding author. Tel.: +44 161 275 6139; fax: +44 161 275 6204.
E-mail addresses: pan@cs.man.ac.uk (J. Pan), horrocks@cs.man.ac.uk (I. Horrocks).

[1] A widely discussed example would be the 'BigWheel' example discussed in, e.g., http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0061.html.

[2] More details of XML Schema Datatypes can be found in Section 3.1.

1570-8268/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2005.08.001

datatypes and, most importantly, guarantees the computability of the kinds of customised datatypes it supports.
2. It should overcome other important limitations of OWL datatyping, such as the absence of negated datatypes and the un-intuitive semantics for unsupported datatypes (which will be further explained in Section 4).
3. It should satisfy the *small extension requirement*, which is two folded: on the one hand, the extension should be a substantial and necessary extension that overcomes the above mentioned limitations of OWL datatyping; on the other hand, following W3C's 'one *small* step at a time' strategy, it should only be as large as is necessary in order to satisfy the requirements.
4. It should be a decidable extension of OWL DL.

This paper makes two main contributions. Firstly, it provides an overview of relevant (to OWL) datatype formalisms, namely those of XML, RDF and OWL itself. Secondly, and most importantly, it presents an extension of OWL DL,[3] called OWL with unary datatype Expressions (OWL-Eu), which satisfies the above requirements.

The rest of the paper is organised as follows. Section 2 briefly introduces the OWL Web Ontology Language. Section 3 describes OWL-related datatype formalisms. Section 4 summarises the limitations of OWL datatyping. Section 5 presents the OWL-Eu language, showing how it satisfies the above four requirements. Section 6 describes some related work, and Section 7 concludes the paper and suggests some future work.

2. An overview of OWL

OWL is a standard (W3C recommendation) for expressing ontologies in the Semantic Web. The OWL language facilitates greater machine understandability of Web resources than that supported by RDFS by providing additional constructors for building class and property descriptions (vocabulary) and new axioms (constraints), along with a formal semantics. The OWL recommendation actually consists of three languages of increasing expressive power: OWL Lite, OWL DL and OWL Full. *OWL Lite* and *OWL DL* are, like DAML + OIL, basically very expressive Description Logics (DLs); they are almost[4] equivalent to the $\mathcal{SHIF}(\mathbf{D}^+)$ and $\mathcal{SHOIN}(\mathbf{D}^+)$ DLs. *OWL Full* provides the same set of constructors as OWL DL, but allows them to be used in an unconstrained way (in the style of RDF). It is easy to show that OWL Full is undecidable, because it does not impose restrictions on the use of transitive properties [12]; therefore, when we mention OWL in this paper, we usually mean OWL DL.

Let $\mathbf{C}$, $\mathbf{R_I}$, $\mathbf{R_D}$ and $\mathbf{I}$ be the sets of URIrefs that can be used to denote concepts, *individual-valued* properties, *data-valued* properties and individuals respectively. An OWL DL *interpretation* is a tuple $\mathcal{I} = (\Delta^{\mathcal{I}}, \Delta_{\mathrm{D}}, \cdot^{\mathcal{I}}, \cdot^{\mathrm{D}})$ where the individual domain $\Delta^{\mathcal{I}}$ is a nonempty set of individuals, the datatype domain Δ_{D} is a nonempty set of data values, $\cdot^{\mathcal{I}}$ is an individual interpretation function that maps

- each individual name $\mathsf{a} \in \mathbf{I}$ to an element $\mathsf{a}^{\mathcal{I}} \in \Delta^{\mathcal{I}}$,
- each concept name $\mathsf{CN} \in \mathbf{C}$ to a subset $\mathsf{CN}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}_{\simeq}}$,
- each *individual-valued* property name $RN \in \mathbf{R_I}$ to a binary relation $RN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and
- each *data-valued* property name $TN \in \mathbf{R_D}$ to a binary relation $TN^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathrm{D}}$,

and $\cdot^{\mathrm{D}}$ is a datatype interpretation function. More details of Δ_{D} and $\cdot^{\mathrm{D}}$ will be presented in Section 3.3.

Let $RN \in \mathbf{R_I}$ an *individual-valued* property URIref, R an *individual-valued* property, $TN \in \mathbf{R_D}$ a *data-valued* property URIref and T a *data-valued* property. Valid OWL DL *individual-valued* properties are defined by the DL syntax:

$$R ::= RN | R^{-};$$

valid OWL DL *data-valued* properties are defined by the DL syntax:

$$T ::= TN.$$

Let $\mathsf{CN} \in \mathbf{C}$ be a concept name, C, D concept descriptions, $o \in \mathbf{I}$ an individual, u an OWL datatype range (cf. Definition 8) and $m \in \mathbb{N}$ an integer. Valid OWL DL concept descriptions are defined by the DL syntax:

$$\begin{aligned} C ::= {} & \top \mid \bot \mid \mathsf{CN} \mid \neg C \mid C \sqcap D \mid C \sqcup D \mid \{o\} \\ & \exists R.C \mid \forall R.C \mid \geqslant mR, \mid \leqslant mR \\ & \exists T.u \mid \forall T.u \mid \geqslant mT, \mid \leqslant mT \end{aligned}$$

The individual interpretation function can be extended to give semantics to concept and property descriptions shown in Table 1, where $\mathsf{A} \in \mathbf{C}$ is a concept URIref, $C, C_1, \ldots, C_n$ are concept descriptions, $S \in \mathbf{R_I}$ is an *individual-valued* property URIref, R is an *individual-valued* property description and $\mathsf{o}, \mathsf{o}_1, \mathsf{o}_2 \in \mathbf{I}$ are individual URIrefs, u is a data range (cf. Definition 8), $T \in \mathbf{R_D}$ is a *data-valued* property and $\sharp$ denotes cardinality.

An OWL DL ontology can be seen as a DL knowledge base [10], which consists of a set of *axioms*, including class axioms, property axioms and individual axioms.[5] Table 2 presents the abstract syntax, DL syntax and semantics of OWL axioms, where $R_1, \ldots, R_n$ are *individual-valued* property descriptions. More details of the semantics of OWL DL can be found in [19].

3. Datatype formalisms

In this section we will provide a brief overview of the XML, RDF and OWL datatype formalisms.

3.1. XML Schema Datatypes

W3C XML Schema Part 2 [4] defines facilities for defining simple types to be used in XML Schema as well as other XML specifications.

Definition 1. An *XML Schema simple type d* is characterised by a value space, $V(d)$, which is a non-empty set, a lexical space,

[3] cf. Section 2 for the differences of three sub-languages of OWL.
[4] They also provide annotation properties, which Description Logics do not.
[5] Individual axioms are also called *facts*.

Table 1
OWL concept and property descriptions

Abstract syntax	DL syntax	Semantics
Class(A)	A	$\mathsf{A}^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
Class(owl:Thing)	$\top$	$\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$
Class(owl:Nothing)	$\bot$	$\bot^{\mathcal{I}} = \emptyset$
intersectionOf($C_1, C_2, \ldots$)	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
unionOf($C_1, C_2, \ldots$)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
complementOf(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
oneOf($\mathsf{o}_1, \mathsf{o}_2, \ldots$)	$\{\mathsf{o}_1\} \sqcup \{\mathsf{o}_2\}$	$(\{\mathsf{o}_1\} \sqcup \{\mathsf{o}_2\})^{\mathcal{I}} = \{\mathsf{o}_1^{\mathcal{I}}, \mathsf{o}_2^{\mathcal{I}}\}$
restriction(R someValuesFrom(C))	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.\langle x, y\rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restriction(R allValuesFrom(C))	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
restriction(R hasValue(o))	$\exists R.\{\mathsf{o}\}$	$(\exists R.\{\mathsf{o}\})^{\mathcal{I}} = \{x \mid \langle x, \mathsf{o}^{\mathcal{I}}\rangle \in R^{\mathcal{I}}\}$
restriction(R minCardinality(m))	$\geqslant mR$	$(\geqslant mR)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \geq m\}$
restriction(R maxCardinality(m))	$\leqslant mR$	$(\leqslant mR)^{\mathcal{I}} = \{x \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq m\}$
restriction(T someValuesFrom(u))	$\exists T.u$	$(\exists T.u)^{\mathcal{I}} = \{x \mid \exists t.\langle x, t\rangle \in T^{\mathcal{I}} \wedge t \in u^{\mathrm{D}}\}$
restriction(T allValuesFrom(u))	$\forall T.u$	$(\forall T.u)^{\mathcal{I}} = \{x \mid \exists t.\langle x, t\rangle \in T^{\mathcal{I}} \rightarrow t \in u^{\mathrm{D}}\}$
restriction(T hasValue(w))	$\exists T.\{w\}$	$(\exists T.\{w\})^{\mathcal{I}} = \{x \mid \langle x, w^{\mathrm{D}}\rangle \in T^{\mathcal{I}}\}$
restriction(T minCardinality(m))	$\geqslant mT$	$(\geqslant mT)^{\mathcal{I}} = \{x \mid \sharp\{t \mid \langle x, t\rangle \in T^{\mathcal{I}}\} \geq m\}$
restriction(T maxCardinality(m))	$\leqslant mT$	$(\leqslant mT)^{\mathcal{I}} = \{x \mid \sharp\{t \mid \langle x, t\rangle \in T^{\mathcal{I}}\} \leq m\}$
ObjectProperty(S)	S	$S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
ObjectProperty(S' inverseOf(S))	S^{-}	$(S^{-})^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
DatatypeProperty(T)	T	$T^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathrm{D}}$

$L(d)$, which is a non-empty set of Unicode [6] strings, and a set of facets, $F(d)$, each of which characterizes a value space along independent axes or dimensions.

XML Schema simple types are divided into disjoint built-in simple types and derived simple types. Derived datatypes can be defined by derivation from primitive or existing derived datatypes by the following three means:

- Derivation by *restriction*, i.e., by using facets on an existing type, so as to limit the number of possible values of the derived type.
- Derivation by *union*, i.e., to allow values from a list of simple types.
- Derivation by *list*, i.e., to define the list type of an existing simple type.

Table 2
OWL axioms

Abstract syntax	DL syntax	Semantics
Class(A partial $C_1 \ldots C_n$)	$\mathsf{A} \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$	$\mathsf{A}^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
Class(A complete $C_1 \ldots C_n$)	$\mathsf{A} \equiv C_1 \sqcap \ldots \sqcap C_n$	$\mathsf{A}^{\mathcal{I}} = C_1^{\mathcal{I}} \cap \ldots \cap C_n^{\mathcal{I}}$
EnumeratedClass(A$\mathsf{o}_1 \ldots \mathsf{o}_n$)	$\mathsf{A} \equiv \{\mathsf{o}_1\} \sqcup \ldots \sqcup \{\mathsf{o}_n\}$	$\mathsf{A}^{\mathcal{I}} = \{\mathsf{o}_1^{\mathcal{I}}, \ldots, \mathsf{o}_n^{\mathcal{I}}\}$
SubClassOf(C_1, C_2)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
EquivalentClasses($C_1 \ldots C_n$)	$C_1 \equiv \ldots \equiv C_n$	$C_1^{\mathcal{I}} = \ldots = C_n^{\mathcal{I}}$
DisjointClasses($C_1 \ldots C_n$)	$C_i \sqsubseteq \neg C_j, (1 \leq i < j \leq n)$	$C_1^{\mathcal{I}} \cap C_n^{\mathcal{I}} = \emptyset, (1 \leq i < j \leq n)$
SubPropertyOf(R_1, R_2)	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
EquivalentProperties($R_1 \ldots R_n$)	$R_1 \equiv \ldots \equiv R_n$	$R_1^{\mathcal{I}} = \ldots = R_n^{\mathcal{I}}$
ObjectProperty(R super(R_1) ... super(R_n)	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$
domain (C_1) ... domain(C_k)	$\geqslant 1R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
range (C_1) ... range(C_h)	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$
[Symmetric]	$R \equiv R^{-}$	$R^{\mathcal{I}} = (R^{-})^{\mathcal{I}}$
[Functional]	Func(R)	$\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in R^{\mathcal{I}}\} \leq 1\}$
[InverseFunctional]	Func(R^{-})	$\{\langle x, y\rangle \mid \sharp\{y.\langle x, y\rangle \in (R^{-})^{\mathcal{I}}\} \leq 1\}$
[Transitive])	Trans(R)	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$
AnnotationProperty(R)		
Individual(o type(C_1) ... type(C_n))	$\mathsf{o} : C_i, 1 \leq i \leq n$	$\mathsf{o}^{\mathcal{I}} \in C_i^{\mathcal{I}}, 1 \leq i \leq n$
value(R_1, o_1) ... value(R_n, o_n)	$\langle \mathsf{o}, \mathsf{o}_i\rangle : R_i, 1 \leq i \leq n$	$\langle \mathsf{o}^{\mathcal{I}}, \mathsf{o}_i^{\mathcal{I}}\rangle \in R_i^{\mathcal{I}}, 1 \leq i \leq n$
SameIndividual($\mathsf{o}_1 \ldots \mathsf{o}_n$)	$\mathsf{o}_1 = \ldots = \mathsf{o}_n$	$\mathsf{o}_1^{\mathcal{I}} = \ldots = \mathsf{o}_n^{\mathcal{I}}$
DifferentIndividuals($\mathsf{o}_1 \ldots \mathsf{o}_n$)	$\mathsf{o}_i \neq \mathsf{o}_j, 1 \leq i < j \leq n$	$\mathsf{o}_i^{\mathcal{I}} \neq \mathsf{o}_j^{\mathcal{I}}, 1 \leq i < j \leq n$

The `atLeast18` datatype defined in Example 1 is a derived simple type (of the base datatype xsd:integer), the value space of which is restricted to integers that are greater than or equal to 18 using the facet minInclusive. The `cameraPrice` datatype defined in Example 4 is a derived simple type by union.

Details of XML Schema derived simple types by list and complex types can be found in [4]. As they are not consistent with the RDF datatype model to be presented in the next section, they are out of the scope of this paper.

3.2. Datatypes in RDF

According to [8], RDF allows the use of datatypes defined by any external type systems, e.g., the XML Schema type system, which conform to the following specification.

Definition 2. A *datatype d* is characterised by a lexical space, $L(d)$, which is an non-empty set of Unicode strings; a value space, $V(d)$, which is an non-empty set, and a total mapping $L2V(d)$ from the lexical space to the value space.

This specification allows the use of non-list XML Schema built-in simple types as datatypes in RDF, although some built-in XML Schema datatypes are problematic because they do not fit the RDF datatype model.[6] Furthermore, comparisons between Definitions 1 and 2 show that RDF does not take XML Schema facets into account, which are essential to define derived simple types.

In RDF, data values are represented by literals.

Definition 3. All *literals* have a lexical form being a Unicode string. *Typed literals* are of the form "s"^^u, where s is a Unicode string, called the *lexical form* of the typed literal, and u is a datatype URI reference. *Plain literals* have a lexical form and optionally a *language tag* as defined by [1], normalised to lowercase.

Example 2. Boolean is a datatype with value space {*true*, *false*}, lexical space {"true", "false", "1", "0"} and lexical-to-value mapping {"true" $\mapsto$ *true*, "false" $\mapsto$ *false*, "1" $\mapsto$ *true*, "0" $\mapsto$ *false*}. "true"^^xsd:boolean is a typed literal, while "true" is a plain literal.

The associations between datatype URI references (e.g., xsd:boolean) and datatypes (e.g., boolean) can be provided by datatype maps defined as follows.

Definition 4. A *datatype map* $\mathbf{M}_d$ is a partial mapping from datatype URI references to datatypes.

Note that XML Schema derived simple types are *not* RDF datatypes because XML Schema provides no mechanism for using URI references to refer to derived simple types.

The semantics of RDF datatypes are defined in terms of $\mathbf{M}_d$-interpretations, which extend RDF-interpretations and RDFS-interpretations (cf. RDF Semantics [8]) with extra conditions for datatypes.

[6] Readers are referred to [8] for more details.

Definition 5. Given a datatype map $\mathbf{M}_d$, an *RDFS* $\mathbf{M}_d$-*interpretation* I of a vocabulary **V** (a set of URIrefs and plain literals) is any RDFS-interpretation of $\mathbf{V} \cup \{u | \exists d.\langle u, d\rangle \in \mathbf{M}_d\}$ which introduces

- a non-empty set **IR** of resources, called the *domain* (or *universe*) of I,
- a set **IP** (the RDF-interpretation requires **IP** to be a sub-set of **IR**) called the *set of properties* in I,
- a set **IC** (the RDFS-interpretation requires **IC** to be a sub-set of **IR**) called the *set of classes* in I, and
- a distinguished subset **LV** of **IR**, called the *set of literal values*, which contains all the plain literals in **V**,
- a mapping *IS* from URIrefs in **V** to **IR**,
- a mapping *IEXT*, called the *extension function*, from **IP** to the powerset of $\mathbf{IR} \times \mathbf{IR}$,
- a mapping *ICEXT*, called the *class extension function*, from **IC** to the set of subsets of **IR**,
- a mapping *IL* from typed literals in **V** into **IR**,

and satisfies the following extra conditions:

1. $\mathbf{LV} = ICEXT(IS(\text{rdfs:Literal}))$,
2. for each plain literal pl, $IL(pl) = pl$,
3. for each pair $\langle u, d\rangle \in \mathbf{M}_d$,
   (a) $ICEXT(d) = V(d) \subseteq \mathbf{LV}$,
   (b) there exist $d \in \mathbf{IR}$ s.t. $IS(u) = d$,
   (c) $IS(u) \in ICEXT(IS(\text{rdfs:Datatype}))$,
   (d) for "s"^^$u' \in \mathbf{V}$, $IS(u') = d$, if $s \in L(d)$, then $IL(\text{"}s\text{"}\hat{}\hat{}u') = L2V(d)(s)$, otherwise, $IL(\text{"s"}\hat{}\hat{}u') \in \mathbf{IR} \setminus \mathbf{LV}$,
4. if $d \in ICEXT(IS(\text{rdfs:Datatype}))$, then $\langle d, IS(\text{rdfs:Literal})\rangle \in IEXT(\text{rdfs:subClassOf})$.

According to Definition 5, **LV** is a subset of **IR**, i.e., literal values are resources. Condition 1 ensures that the class extension of rdfs:Literal is **LV**. Condition 2 ensures that the plain literals are interpreted as themselves, and that **LV** contains interpretations of all valid typed literals of datatypes in $\mathbf{M}_d$. Condition 3a asserts that RDF(S) datatypes are classes (because datatypes are interpreted using the class extension function *ICEXT*), Condition 3b ensures that there is a resource d for datatype d in $\mathbf{M}_d$, and Condition 3c ensures that the class rdfs:Datatype contains the datatypes used in any satisfying $\mathbf{M}_d$-interpretation. Condition 3d explains why the range of *IL* is **IR** rather than **LV** (because, for "s"^^u, if $s \notin L(IS(u))$, then $IL(\text{"s"}\hat{}\hat{}u) \notin \mathbf{LV}$); note that this is different from OWL datatypes (cf. Definition 9). Condition 4 requires that RDF(S) datatypes are *sub-classes* of rdfs:Literal.

3.3. Datatypes in OWL

OWL datatyping adopts the RDF specification of datatypes and data values. It extends RDF datatyping by (i) allowing different OWL reasoners to provide different supported datatypes, and (ii) introducing the use of so called enumerated datatypes.

Definition 6. Given a datatype map $\mathbf{M}_d$, a datatype URI reference u is called a *supported datatype URI reference w.r.t* $\mathbf{M}_d$ if there exists a datatype d s.t. $\mathbf{M}_d(u) = d$ (in this case, d is called a *supported datatype w.r.t.* $\mathbf{M}_d$); otherwise, u is called an *unsupported datatype URI reference w.r.t.* $\mathbf{M}_d$.

Definition 7. Let $y_1, \ldots, y_n$ be typed literals. An *enumerated datatype* is of the form oneOf($y_1, \ldots, y_n$).

The kinds of datatypes provided by OWL are called OWL data ranges, which can be used in datatype-related class descriptions. In fact, in line (13) and (14) of Table 1, u is an OWL data range.

Definition 8. An *OWL data range* has one of the forms: (i) a datatype URI reference, (ii) an enumerated datatype, or (iii) rdf:Literal.

The semantics of OWL DL datatypes are defined in terms of OWL datatype interpretations.

Definition 9. An *OWL datatype interpretation* w.r.t. to a datatype map $\mathbf{M}_d$ is a pair $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$, where the datatype domain $\Delta_{\mathrm{D}} = \mathbf{PL} \cup \bigcup_{\text{for each supported datatype URIref u w.r.t. } \mathbf{M}_p} V(\mathbf{M}_p(u))$ (**PL** is the value space for plain literals, i.e., the union of the set of Unicode strings and the set of pairs of Unicode strings and language tags) and $\cdot^{\mathrm{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. rdfs:Literal$^{\mathrm{D}} = \Delta_{\mathrm{D}}$;
2. for each plain literal l, $l^{\mathrm{D}} = l \in \mathbf{PL}$;
3. for each supported datatype URIref u (let $d = M_d(u)$):
   (a) $u^{\mathrm{D}} = V(d) \subseteq \Delta_{\mathrm{D}}$,
   (b) if $s \in L(d)$, then ("s"^^u)$^{\mathrm{D}} = L2V(d)(s)$,
   (c) if $s \notin L(d)$, then ("s"^^u)$^{\mathrm{D}}$ is not defined;
4. for each unsupported datatype URIref u, $u^{\mathrm{D}} \subseteq \Delta_{\mathrm{D}}$, and ("s"^^$u$)$^{\mathrm{D}} \in u^{\mathrm{D}}$.
5. each enumerated datatype oneOf($y_1, \ldots, y_n$) is interpreted as $y_1^{\mathrm{D}} \cup \ldots \cup y_n^{\mathrm{D}}$.

The above definition shows that OWL datatyping is similar to RDF datatyping, except that (i) RDF datatypes are classes, while OWL DL datatypes are not classes,[7] and (ii) in RDF ill-defined typed literals are interpreted as resources in $\mathbf{IR} \setminus \mathbf{LV}$, while in OWL DL the interpretation of ill-defined typed literals are undefined.

4. Limitations of OWL datatyping

OWL datatyping has the following serious limitations, which discourage potential users from adopting OWL DL in their SW and ontology applications [16,22].

1. OWL does not support customised datatypes (except enumerated datatypes). Firstly, XML Schema derived simple types are not OWL DL datatypes, because of the problem of datatype URI references for XML Schema derived simple types. Secondly, OWL does not provide a mechanism to tell which (customised) datatypes can be used together so that the language is still decidable.
2. OWL does not support negated datatypes. For example, 'all integers but 0', which is the relativised negation of the enumerated datatype oneOf("0"^^xsd:integer), is not expressible in OWL. Moreover, negated datatypes are *necessary* in the negated normal form (NNF)[8] of datatype-related class descriptions in, e.g., DL tableaux algorithms.
3. An OWL DL datatype domain seriously restricts the interpretations of typed literals with unsupported datatype URIrefs. According to Definition 9, datatype domain is equal to the set of all plain literals together with the value spaces of all supported datatypes. For example, given the datatype map $\mathbf{M}_{d1} = \{$xsd:integer $\mapsto$ *integer*, xsd:string $\mapsto$ *string*$\}$, "1.278e-3"^^xsd:float has to be interpreted as either an integer, a string or a string with a language tag, which is counter-intuitive.

5. OWL-Eu

This section presents OWL-Eu and elaborates how OWL-Eu satisfies the four requirements (listed in Section 1) in the following four sub-sections.

5.1. Supporting Customised Datatypes

OWL-Eu supports customised datatypes through unary datatype expressions based on unary datatype groups. Intuitively, an unary datatype group extends the OWL datatyping with a hierarchy of supported datatypes.[9]

Definition 10. A *unary datatype group* $\mathcal{G}$ is a triple ($\mathbf{M}_d$, $\mathbf{B}$, dom), where $\mathbf{M}_d$ is the *datatype map* of $\mathcal{G}$, $\mathbf{B}$ is the set of *primitive base datatype* URI references in $\mathcal{G}$ and dom is the *declared domain function*. We call $\mathbf{S}$ the set of supported datatype URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_d(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. We assume that there exists a unary datatype URI reference owlx:DatatypeBottom $\notin \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$, if $u \in \mathbf{B}$, dom(u) $= u$; otherwise, dom(u) $= v$, where $v \in \mathbf{B}$.

Definition 10 ensures that all the primitive base datatype URIrefs of $\mathcal{G}$ are supported ($\mathbf{B} \subseteq \mathbf{S}$) and that each supported datatype URIref relates to a primitive base datatype URIref through the declared domain function dom.

Example 3. $\mathcal{G}_1 = (\mathbf{M}_{d1}, \mathbf{B}_1, \mathsf{dom}_1)$ is a unary datatype group, where

- $\mathbf{M}_{d1} = \{$xsd:integer $\mapsto$ *integer*, xsd:string $\mapsto$ *string*, xsd:nonNegativeInteger $\mapsto \geq_0$, xsdx:integerLessThanN $\mapsto <_N\}$,
- $\mathbf{B}_1 = \{$xsd:string, xsd:integer$\}$, and

[7] In fact, classes and datatypes in OWL DL use different interpretation functions; cf. Section 2.

[8] A concept is in negation normal form iff negation is applied only to atomic concept names, nominals or datatypes.

[9] Note that in [16] datatype groups allow arbitrary datatype predicates, while here we consider only datatypes, which can be regarded as *unary* datatype predicates.

- $\mathrm{dom}_1 = \{$xsd:integer $\mapsto$ xsd:integer, xsd:string $\mapsto$ xsd:string, xsd:nonNega tiveInteger $\mapsto$ xsd:integer, xsdx:integerLessThanN $\mapsto$ xsd:integer$\}$.

According to $\mathbf{M}_{d1}$, we have $\mathbf{S}_1 = \{$xsd:integer, xsd:string, xsd:nonNegativeInteger, xsdx:integer LessThanN$\}$, hence $\mathbf{B}_1 \subseteq \mathbf{S}_1$. Note that the value space of $<_N$ is

$$V(<_N) = \{i \in V(\text{integer}) | i < L2V(\text{integer})(N)\},$$

and by $<_N$ we mean there exists a supported datatype $<_N$ for each integer $L2V(integer)(N)$.

Based on a unary datatype group, OWL-Eu provides a formalism (called datatype expressions) for constructing customised datatypes using supported datatypes.

Definition 11. Let $\mathcal{G}$ be a unary datatype group. The set of *$\mathcal{G}$-unary datatype expressions* in abstract syntax (corresponding DL syntax can be found in Table 3), abbreviated **Dexp**$\mathcal{G}$, is inductively defined as follows:

1. *atomic expressions* $u \in$ **Dexp** $(\mathcal{G})$, for a datatype URIref u;
2. *relativised negated expressions* not$(u) \in$ **Dexp** $(\mathcal{G})$, for a datatype URIref u;
3. *enumerated expressions* oneOf$(l_1, \ldots, l_n) \in$ **Dexp** $(\mathcal{G})$, for literals $l_1, \ldots, l_n$;
4. *conjunctive expressions* and$(E_1, \ldots, E_n) \in$ **Dexp** $(\mathcal{G})$, for datatype expressions $E_1, \ldots, E_n \in$ **Dexp** $(\mathcal{G})$;
5. *disjunctive expressions* or$(E_1, \ldots, E_n) \in$ **Dexp** $(\mathcal{G})$, for datatype expressions $E_1, \ldots, E_n \in$ **Dexp** $(\mathcal{G})$.

Example 4. $\mathcal{G}$-unary datatype expressions can be used to represent XML Schema non-list simple types. Given the unary datatype group $\mathcal{G}_1$ presented in Example 3,

built-in XML Schema simple types *integer, string, nonNegativeInteger* are supported datatypes in $\mathcal{G}_1$;

the XML Schema derived simple type (using only one facet) `atLeast18` defined in Example 1 can be represented by the relativised negated expression

not(xsdx:integerLessThan18);

the following XML Schema derived simple type (using more than one facet) `humanAge`

```
<simpleType name=''humanAge''>
   <restriction base=''xsd:integer''>
      <minInclusive value=''0''/>
      <maxExclusive value=''150''/>
   </restriction>
</simpleType>
```

can be represented by the following conjunctive expression

and(xsd:nonNegativeInteger, xsdx:integerLessThan150);

the following XML Schema derived union simple type

```
<simpleType name=''cameraPrice''>
   <union>
      <simpleType>
         <restriction base=''xsd:nonNegativeInteger''>
           <maxExclusive value=''100000''/>
         </restriction>
      </simpleType>
      <simpleType>
         <restriction base=''xsd:string''>
           <enumeration value=''low''/>
           <enumeration value=''medium''/>
           <enumeration value='èxpensive''/>
         </restriction>
      </simpleType>
   </union>
</simpleType>
```

can be represented by the following disjunctive expression

```
or(
    and(xsd:nonNegativeInteger, xsdx:integerLessThan100000)
    oneOf("low"^^xsd:string, "medium"^^xsd:string, "expensive"^^xsd:string)
                                    ).
```

Table 3
Syntax and semantics of datatype expressions (OWL-Eu data ranges)

Abstract syntax	DL syntax	Semantics
a datatype URIref u	u	u^{D}
oneOf($l_1, \ldots, l_n$)	$\{l_1, \ldots, l_n\}$	$\{l_1^{\mathrm{D}}\} \cup \ldots \cup \{l_n^{\mathrm{D}}\}$
not(u)	$\overline{u}$	$(\mathsf{dom}(u))^{\mathrm{D}} \setminus u^{\mathrm{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}\Delta_{\mathrm{D}} \setminus u^{\mathrm{D}}$ otherwise
and($E_1, \ldots, E_n$)	$E_1 \wedge \ldots \wedge E_n$	$E_1^{\mathrm{D}} \cap \ldots \cap E_n^{\mathrm{D}}$
or(P, Q)	$E_1 \vee \ldots \vee E_n$	$E_1^{\mathrm{D}} \cup \ldots \cup E_n^{\mathrm{D}}$

We now define the interpretation of a unary datatype group.

Definition 12. A *datatype interpretation* $\mathcal{I}_{\mathrm{D}}$ of a unary datatype group $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ is a pair $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$, where Δ_{D} (the datatype domain) is a non-empty set and $\cdot^{\mathrm{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

1. $(\text{rdfs:Literal})^{\mathrm{D}} = \Delta_{\mathrm{D}}$ and $(\mathsf{owlx{:}DatatypeBottom})^{\mathrm{D}} = \emptyset$;
2. for each plain literal l, $l^{\mathrm{D}} = l \in \mathbf{PL}$ and $\mathbf{PL} \subseteq \Delta_{\mathrm{D}}$;[10]
3. for any two primitive base datatype URIrefs $u_1, u_2 \in \mathbf{B}$: $u_1^{\mathrm{D}} \cap u_2^{\mathrm{D}} = \emptyset$;
4. for each supported datatype URIref $u \in \mathbf{S}$, where $d = \mathbf{M}_d(u)$:
   (a) $u^{\mathrm{D}} = V(d) \subseteq \Delta_{\mathrm{D}}$, $L(u) \subseteq L(\mathsf{dom}(u))$ and $L2V(u) \subseteq L2V(\mathsf{dom}(u))$;
   (b) if $s \in L(d)$, then $(\text{“s”}\hat{}\hat{}u)^{\mathrm{D}} = L2V(d)(s)$; otherwise, $(\text{“s”}\hat{}\hat{}u)^{\mathrm{D}}$ is not defined;
5. $\forall u \notin \mathbf{S}$, $u^{\mathrm{D}} \subseteq \Delta_{\mathrm{D}}$, and $\text{“s”}\hat{}\hat{}u \in u^{\mathrm{D}}$.

Moreover, we extend $\cdot^{\mathrm{D}}$ to $\mathcal{G}$ unary datatype expression as shown in Table 3. Let E be a $\mathcal{G}$ unary datatype expression, the negation of E is of the form $\neg E$, which is interpreted as $\Delta_{\mathrm{D}} / E^{\mathrm{D}}$.

In Definition 12, Condition 3 ensures that the value spaces of all primitive base datatypes are disjoint with each other. Condition 4a ensures that each supported datatype is a derived datatype of its primitive base datatype. Please note the difference between a relativised negated expression and the negation of a unary datatype expression: the former one is a kind of unary datatype expression, while the latter one is the form of negation of all kinds of unary datatype expressions. Furthermore, Definition 12 indicates enumerated expressions are special forms of disjunctive expressions.

It is worth noting that the (full) negation of a unary datatype expression is also a unary datatype expression. This can be easily shown as follows.

- $\neg u$: if $u \in \mathbf{B}$, $\neg u = \overline{u}$; otherwise, $\neg u = \overline{u} \vee \overline{\mathsf{dom}(u)}$.
- $\neg \overline{u}$: if $u \in \mathbf{B}$, $\neg \overline{u} = u$; otherwise, $\neg \overline{u} = u \vee \overline{\mathsf{dom}(u)}$.
- $\neg(u_1 \wedge \ldots \wedge u_n) = \neg u_1 \vee \ldots \vee \neg u_n$.
- $\neg(u_1 \vee \ldots \vee u_n) = \neg u_1 \wedge \ldots \wedge \neg u_n$.

Next, we introduce the kind of basic reasoning mechanisms required for a unary datatype group.

[10] **PL** is the value space for plain literals; cf. Definition 9.

Definition 13. Let $\mathbf{V}$ be a set of variables, $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ a unary datatype group and $u \in \mathbf{B}$ a primitive base datatype URIref. A datatype conjunction of u is of the form

$$\mathcal{C} = \bigwedge_{j=1}^{k} u_j(v_j) \wedge \bigwedge_{i=1}^{l} \neq_i (v_1^{(i)}, v_2^{(i)}), \tag{1}$$

where the v_j are variables from $\mathbf{V}$, $v_1^{(i)}$, $v_2^{(i)}$ are variables appear in $\bigwedge_{j=1}^{k} u_j(v_j)$, u_j are datatype URI references from $\mathbf{S}$ such that $\mathsf{dom}(u_j) = u$, and $\neq_i$ are the inequality predicates for primitive base datatypes $\mathbf{M}_d(\mathsf{dom}(u_i))$ where u_i appear in $\bigwedge_{j=1}^{k} u_j(v_j)$.

A datatype conjunction $\mathcal{C}$ is called *satisfiable* iff there exist an interpretation $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of $\mathcal{G}$ and a function δ mapping the variables in $\mathcal{C}$ to data values in Δ_{D} s.t. $\delta(v_j) \in u_j^{\mathrm{D}}$ (for all $1 \leq j \leq k$) and $\{\delta(v_1^{(i)}), \delta(v_2^{(i)})\} \subseteq u_i^{\mathrm{D}}$ and $\delta(v_1^{(i)}) \neq \delta(v_2^{(i)})$ (for all $1 \leq i \leq l$). Such a function δ is called a *solution* for $\mathcal{C}$ w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$.

We end this section by elaborating the conditions that computable unary datatype groups require.

Definition 14. A unary datatype group $\mathcal{G}$ is *conforming* iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exist $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathrm{D}} = \overline{u}^{\mathrm{D}}$, and
2. for each primitive base datatype in $\mathcal{G}$, the satisfiability problems for finite datatype conjunctions of the form (1) is decidable.

5.2. Small extension: from OWL DL to OWL-Eu

In this section, we present a small extension of OWL DL, i.e., OWL-Eu. The underpinning DL of OWL-Eu is $\mathcal{SHOIN}(\mathcal{G}_1)$, i.e., the $\mathcal{SHOIN}$ DL combined with a unary datatype group $\mathcal{G}$ (1 for unary). Specifically, OWL-Eu (only) extends OWL data range (cf. Definition 8) to OWL-Eu data ranges defined as follows.

Definition 15. An *OWL-Eu data range* is a $\mathcal{G}$ unary datatype expression. Abstract (as well as DL) syntax and model-theoretic semantics of OWL-Eu data ranges are presented in Table 3.

The consequence of the extension is that customised datatypes, represented by OWL-Eu data ranges, can be used in datatype exists restrictions ($\exists T.u$) and datatype value restrictions ($\forall T.u$), where T is a datatype property and u is an OWL-Eu data range (cf. Table 1). Hence, this extension of OWL DL is as large as is necessary to support customised datatypes.

Example 5. PCs with memory size greater than or equal to 512 Mb and with price cheaper than 700 pounds can be represented in the following OWL-Eu concept description in DL syntax (cf. Table 3):

$$\mathsf{PC} \sqcap \exists \mathit{memorySizeInMb}.\overline{<_{512}} \sqcap \exists \mathit{priceInPound}.<_{700},$$

where $\overline{<_{512}}$ is a relativised negated expression and $<_{700}$ is a supported datatype in $\mathcal{G}_1$.

5.3. Decidability of OWL-Eu

Now we show that OWL-Eu is decidable by showing $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability w.r.t. knowledge bases. To decide $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability and subsumption problem w.r.t. knowledge bases, a DL reasoner can use a datatype reasoner to answer datatype queries. Intuitively, a datatype query is a disjunction of datatype expression conjunctions, possibly together with some equality and inequality constraints.

Definition 16. (Datatype Query) For a unary datatype group $\mathcal{G}_1$, a *datatype query* is of the form

$$\mathcal{Q} := \bigvee_{j=1}^{k} \mathcal{C}_{d_j} \wedge \bigwedge_{j_1=1}^{k_1} \neq (v_{(j_1,1)}, v_{(j_1,2)}) \wedge \bigvee_{j_2=1}^{k_2} = (v_{(j_2,1)}, \ldots, v_{(j_2,m_{j_2})}), \quad (2)$$

where $\mathcal{C}_{d_j}$ is a (possibly negated) unary datatype expression conjunction, $v_{(s)}$ are variables appearing in $\mathcal{C}_{d_1}, \ldots, \mathcal{C}_{d_k}$, and $\neq$ and $=$ are called the *value inequality predicate* and *value equality predicate*, respectively. A datatype query is *satisfiable* iff there exists an interpretation $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of $\mathcal{G}_1$ and a function δ mapping the variables in $\mathcal{C}_{d_1}, \ldots, \mathcal{C}_{d_k}$ to data values in Δ_{D} s.t.

δ is a solution for one of $\mathcal{C}_{d_1}, \ldots, \mathcal{C}_{d_k}$ w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ and, $\delta(v_{(j_1,1)}) \neq \delta(v_{(j_1,2)})$ for all $1 \leq j_1 \leq k_1$,[11] there exist some $j_2 (1 \leq j_2 \leq k_2)$ s.t. $\delta(v_{(j_2,1)}) = \cdots = \delta(v_{(j_2,m_{j_2})})$.

Such a function δ is called a *solution* for $\mathcal{Q}$ w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$.

Lemma 1. *For $\mathcal{G}$ a conforming unary datatype group, datatype queries of the form (2) are decidable.*

Proof. Firstly, we will show that the satisfiability problem of (possibly negated) $\mathcal{G}$-datatype expression conjunctions is decidable. It is trivial to reduce the satisfiability problem for $\mathcal{G}$-datatype expression conjunctions to the satisfiability problem for predicate conjunctions over $\mathcal{G}$:

1. Due to Condition 1 of a conforming unary datatype group (cf. Definition 14), we can trivially eliminate relativised negated expressions. Similarly, their (full) negations can be reduced as follows:
$$\neg\overline{u_i}(v_i) \equiv \begin{cases} u_i(v_i) \vee \overline{\mathsf{dom}}(u_i)(v_i) & \text{if } u_i \in \mathbf{S} \setminus \mathbf{B}, \\ u_i(v_i) & \text{otherwise,} \end{cases}$$
according to Definition 12.
2. The `and` and `or` constructors simply introduce disjunctions of datatype conjunctions of $\mathcal{G}$. Due to Condition 3 of Definition 12, datatype conjunctions are unsatisfiable if there exist variables shared among supported datatypes derived from different primitive based datatypes. Therefore, datatype conjunctions of $\mathcal{G}$ can be reduced to datatype conjunctions of primitive base datatypes. According to Definition 14, the satisfiability problem of datatype conjunctions of primitive base datatypes is decidable; therefore, a $\mathcal{G}$-datatype expression conjunction is satisfiable iff one of its disjuncts is satisfiable.

Secondly, we show how to handle the extra constraints introduced by the *value inequality predicate* and *value equality predicate*. We can transform the general equality and inequality constraints into $\mathcal{V}$, a disjunction of conjunctions of the forms $=(v_i, v_j)$ or $\neq(v_{i'}, v_{j'})$. For each satisfiable $\mathcal{G}$-datatype expression conjunction $\mathcal{C}_{E_j}$, we can further extend $\mathcal{C}_{E_j}$ to $\mathcal{C}'_{E_j}$ by adding new conjuncts $=_u (v_i, v_j)$ and/or $\neq_u (v_{i'}, v_{j'})$ into $\mathcal{C}_E$. $\mathcal{Q}$ is *unsatisfiable* if all $\mathcal{C}'_{E_j}$ are *unsatisfiable*; otherwise, $\mathcal{Q}$ is *satisfiable*. □

We will show the decidability of $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability w.r.t. TBoxes and RBoxes by reducing it to the $\mathcal{SHOIQ}$-concept satisfiability w.r.t. TBoxes and RBoxes. The proof is inspired by the proof (Lutz [14, pp. 32–33]) of the decidability of $\mathcal{ALCF}(\mathcal{D})$-concept satisfiability w.r.t. to general TBoxes, where $\mathcal{ALCF}(\mathcal{D})$ is obtained from $\mathcal{ALCF}(\mathcal{D})$ by restricting the concrete domain constructor to concrete features in place of feature chains. The basic idea behind the reduction is that we can replace each datatype group-based concept C in $\mathcal{T}$ with a new atomic primitive concept A_C in $\mathcal{T}'$. We then compute the satisfiability problem for all possible conjunctions of datatype group-based concepts (and their negations) in $\mathcal{T}$ (of which there are only a finite number), and in case a conjunction $C_1 \sqcap \ldots \sqcap C_n$ is unsatisfiable, we add an axiom $A_{C_1} \sqcap \ldots \sqcap A_{C_n} \sqsubseteq \bot$ to $\mathcal{T}'$.

For example, unary datatype group-based concepts $\exists T.>_1$ and $\forall T.\leq_0$ occurring in $\mathcal{T}$ would be replaced with $A_{\exists T.>_1}$ and $A_{\forall T.\leq_0}$ in $\mathcal{T}'$, and $A_{\exists T.>_1} \sqcap A_{\forall T.\leq_0} \sqsubseteq \bot$ would be added to $\mathcal{T}'$ because $\exists T.>_1 \sqcap \forall T.\leq_0$ is *unsatisfiable* (i.e., there is no solution for the predicate conjunction $>_1 (v) \wedge \leq_0 (v)$).

Theorem 1. *The $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability problem w.r.t. a knowledge base is decidable if the combined unary datatype group is conforming.*

Proof. We prove the theorem by reducing $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability w.r.t. a knowledge base to the $\mathcal{SHOIQ}$-concept satisfiability w.r.t. TBoxes and RBoxes. Let D be an $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept for satisfiability checking, $\mathsf{cl}_{\mathcal{T}}(D)$ the set of all the sub-concepts of concepts in $\{D\} \cup \{D_1, D_2 | D_1 \sqsubseteq D_2 \in \mathcal{T} \text{or } D_1 = D_2 \in \mathcal{T}\}$, and $\{C_1, \ldots, C_k\} \subseteq \mathsf{cl}_{\mathcal{T}}(D)$ the set of all the datatype group-based concepts (and their negations) in $\mathsf{cl}_{\mathcal{T}}(D)$, i.e., each C_i $(1 \leq i \leq k)$ is of one of the four forms: $\exists T.d$, $\forall T.d$, $\leqslant nT.d$ and $\geqslant nT.d$, where T is a concrete role name, d is a unary datatype expression and n is an integer. There are two remarks here. Firstly, we assume that $C_1, \ldots, C_k$ are in their negation normal forms; i.e., negations only appear in front of atomic concepts. Secondly, as we have shown in Section 5.1, negations of unary datatype expressions are still unary datatype expressions.

We assume that all the functional concrete role axioms in $\mathcal{R}$ of the form $\mathsf{Func}(T)$ are encoded into concept inclusion axioms of the form $\top \sqsubseteq \leqslant 1T.\top_{\mathrm{D}}$ in $\mathcal{T}$. We assume that all the individual axioms of the form $\mathsf{a} : C$ are encoded into concept inclusion axioms of the form $\{\mathsf{a}\} \sqsubseteq C$, that all the individual axioms of

[11] Note that, if $v = \langle v_1, \ldots, v_n \rangle$, $\delta(v)$ is an abbreviation for $\langle \delta(v_1), \ldots, \delta(v_n) \rangle$.

the form $\langle \mathsf{a}, \mathsf{b} \rangle : R$ are encoded into concept inclusion axioms of the form $\{\mathsf{a}\} \sqsubseteq \exists R.\{\mathsf{b}\}$ and that all the individual axioms of the form $\langle \mathsf{a}, l \rangle : T$ are encoded into concept inclusion axioms of the form $\{\mathsf{a}\} \sqsubseteq \exists T.\{l\}$.

We define a mapping π that maps unary datatype group-based concept conjunctions of the form $S = B_1 \sqcap \ldots \sqcap B_h$, where $\{B_1, \ldots B_h\} \subseteq \{C_1, \ldots, C_k\}$, to a corresponding datatype query $\pi(S)$.

(**Step 1**) For each B_j of the form $\exists T.d$, $\pi(S)$ contains a conjunct $d(v_j^T)$, where each v_j^T is a variable, with the corresponding concrete role name T as its superscript.

(**Step 2**) For each B_j of the form $\geqslant nT.d$, $\pi(S)$ contains a conjunct

$$\bigwedge_{a=1}^{m} d(v_{j_a}^T) \wedge \bigwedge_{1 \leq a < b \leq m} \neq (v_{j_a}^T, v_{j_b}^T)$$

where the inequality constraints are used to make sure the variables $v_{j_1}^T \ldots, v_{j_m}^T$ are mapped to different data values. We will not introduce any more new variables (with superscriptions) in the following steps.

(**Step 3**) For each B_j of the form $\forall T.d$, let A_j be the set of all variables that were introduced in (Step 1) and (Step 2) of the form $v^{T'}$, where the superscript T' matches the corresponding concrete role name T in $\forall T.d$. A variable $v^{T'}$ *matches* a concrete role T if $T' \boxed{\equiv} T$. Then $\pi(S)$ contains a conjunct

$$\bigwedge_{\forall v \in A_j} d(v).$$

(**Step 4**) For each B_j of the form $\leqslant mT_1, \ldots, T_{n_j}.E$, similarly to (Step 3), we can define a set A_j for B_j. Let $|A_j| = m'$. If $m' \leq m$, then let $P(A, x)$ be the function that maps a set A to the set of all the partitions of A with size x; i.e., for each partition $Q = \{q_1, \ldots, q_x\} \in P(A, x)$, $q_1, \ldots, q_n$ are non-empty sets, $q_a \cap q_b = \emptyset$ (for $1 \leq a < b \leq x$) and $A = q_1 \cup \ldots \cup q_x$. Then $\pi(S)$ contains a conjunct

$$\bigvee_{Q \in P(A_j, m)} \bigwedge_{q \in Q} \bigwedge_{v_1, v_2 \in q} d(v_1) \wedge d(v_2) \rightarrow = (v_1, v_2),$$

we can apply the "$x \Rightarrow y \equiv \neg x \vee y$" equivalence and De-Morgan's law to this conjunct to give

$$\bigvee_{Q \in P(A_j, m)} \bigwedge_{q \in Q} \bigwedge_{v_1, v_2 \in q} \neg d(v_1) \vee \neg d(v_2) \vee = (v_1, v_2).$$

Since the satisfiability problem for a datatype query is decidable, for each possible $S = B_1 \sqcap \ldots \sqcap B_h$, where $\{B_1, \ldots, B_h\} \subseteq \{C_1, \ldots, C_k\}$, we can decide if $\pi(S)$ is *satisfiable* or not.

Now we can reduce the $\mathcal{SHOIQ}(\mathcal{G}_1)$-concept satisfiability problem w.r.t. a knowledge base to the $\mathcal{SHOIQ}$-concept satisfiability problem w.r.t. a TBox and an RBox, by introducing some new atomic primitive concepts (to represent C_i, for each $1 \leq i \leq k$) and some concept inclusion axioms about these atomic primitive concepts (to capture all the possible contradictions caused by S) as follows:

(1) We create an atomic primitive concept A_{C_i} for each $C_i \in \{C_1, \ldots, C_k\}$, and transform $\mathcal{T}$ and D into $\mathcal{T}'$ and D' by replacing all C_i with A_{C_i} in $\mathcal{T}$ and D. We transform $\mathcal{R}$ into $\mathcal{R}'$ by removing all the concrete role inclusion axioms.

(2) For each $S = B_1 \sqcap \ldots \sqcap B_h$, where $\{B_1, \ldots, B_h\} \subseteq \{C_1, \ldots, C_k\}$, if $\pi(S)$ is *unsatisfiable*, we add the following concept inclusion axiom into $\mathcal{T}'$:

$$A_{B_1} \sqcap \ldots \sqcap A_{B_h} \sqsubseteq \bot.$$

Claim. (i) *For any $S = B_1 \sqcap \ldots \sqcap B_h$, where $\{B_1, \ldots, B_h\} \subseteq \{C_1, \ldots, C_k\}$, S is satisfiable iff $\pi(S)$ is satisfiable.* (ii) *All the possible contradictions caused by possible datatype group-based sub-concept conjunctions in $\mathsf{cl}_{\mathcal{T}}(D)$ have been encoded in the TBox $\mathcal{T}'$.* (iii) *D is satisfiable w.r.t. $\mathcal{T}$ and $\mathcal{R}$ iff D' is satisfiable w.r.t. $\mathcal{T}'$ and $\mathcal{R}'$.*

Claim (i) is true because the mappings in (Steps 1–4) exactly generate the needed datatype queries $\pi(S)$ according to the semantics of unary datatype group-based concepts.

- (Step 1): For each B_j of the form $\exists T.d$, $\pi(S)$ contains a conjunct $d(v_j^T)$. If $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ is an interpretation of $\mathcal{G}$ and δ is a solution of $d(v_j^T)$ w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of this conjunct, we have $\delta(v_j^T) \in d^{\mathrm{D}}$. Furthermore, the concrete role names T are used in superscripts of the corresponding variables, so as to assure that further constraints from datatype expression value and atmost restrictions can be properly added to these variables.
- (Step 2): For each B_j of the form $\geqslant mT.d$, $\pi(S)$ contains a conjunct $\bigwedge_{a=1}^{m} d(v_{j_a}^T) \wedge \bigwedge_{1 \leq a < b \leq m} \neq (v_{j_a}^T, v_{j_b}^T)$. If $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ is an interpretation of $\mathcal{G}$ and δ is a solution w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of this conjunct, we have $\delta(v_{j_a}^T) \in d^{\mathrm{D}}$ and $\delta(v_{j_a}^T) \neq \delta(v_{j_b}^T)$ for all $1 \leq a < b \leq m$; viz. there are at least m data values that satisfy the unary datatype expression d. The purpose of using superscripts in variables is the same as (Step 1).
- (Step 3): For each B_j of the form $\forall T.d$, $\pi(S)$ contains a conjunct $\bigwedge_{v \in A_j} d(v)$. Since in (Step 1) and (Step 2) we have generated all the needed variables, the set A_j includes all the tuples of variables, the superscripts of which match $T_1, \ldots, T_{n_j}$. If $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ is an interpretation of $\mathcal{G}$ and δ is a solution w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of the above conjunct, we have $\delta(v) \in d^{\mathrm{D}}$, for all $v \in A_j$.
- (Step 4): For each B_j of the form $\leqslant mT.d$, $\pi(S)$ contains a conjunct

$$\bigvee_{Q \in P(A_j, m)} \bigwedge_{q \in Q} \bigwedge_{\rightharpoonup v_1, \rightharpoonup v_2 \in q} d(v_1) \wedge d(v_2) \rightarrow = (v_1, v_2),$$

if $m < |A_j|$. The set A_j is constructed as that in (Step 3), and $P(A_j, m)$ is the set of all the partitions of A_j with size m. If $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ is an interpretation of $\mathcal{G}$ and δ is a solution w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ of this conjunct, there exists a partition Q, s.t. for all $q_i \in Q$ $(1 \leq i \leq m)$, any pairs of variable v_1, v_2 must satisfy that if both $\delta(v_1) \in d^{\mathrm{D}}$ and $\delta(v_2) \in d^{\mathrm{D}}$ are true, then $\delta(v_1) = \delta(v_2)$. In other words, there are at most m different data values that are linked through the concrete roles T and satisfy d.

For claim (ii). Firstly, due to the (1), it is obvious that D' is an $\mathcal{SHOIQ}$-concept and $\mathcal{T}'$ contains no unary datatype group-based concepts, and there are no concrete roles in $\mathcal{R}'$. Secondly, due to (2), claim (i) and that $\mathcal{G}$-datatype queries are decidable, for any possible datatype group-based concept conjunction $S = B_1 \sqcap \ldots \sqcap B_h$ and if $\pi(S)$ is *unsatisfiable*, there is an axiom $A_{B_1} \sqcap \ldots \sqcap A_{B_h} \sqsubseteq \bot$ in $\mathcal{T}'$. Therefore, all the possible contradictions caused by possible datatype group-based sub-concept conjunctions in $\mathsf{cl}_{\mathcal{T}}(D)$ have been encoded in the TBox $\mathcal{T}'$.

For claim (iii). If D is *satisfiable* w.r.t. $\mathcal{T}$ and $\mathcal{R}$, then there is a model $\mathcal{I}$, s.t. $\mathcal{I} \models D, \mathcal{I} \models \mathcal{T}$ and $\mathcal{I} \models \mathcal{R}$. We show how to construct a model $\mathcal{I}'$ of D' w.r.t. $\mathcal{T}'$ and $\mathcal{R}'$ from $\mathcal{I}.\mathcal{I}'$ will be identical to $\mathcal{I}$ in every respect except for concrete roles (there are no concrete roles in $\mathcal{I}'$) and the atomic primitive concepts A_{C_i} for each $C_i \in \{C_1, \ldots, C_k\}$ (there are no A_{C_i} in $\mathcal{I}$). So we only need to construct $A^{\mathcal{I}'}_{C_i} : A^{\mathcal{I}'}_{C_i} = C_i^{\mathcal{I}}$. Due to the constructions of $D', \mathcal{T}', \mathcal{R}'$, we have $D'^{\mathcal{I}'} \neq \emptyset, \mathcal{I}' \models \mathcal{T}'$ and $\mathcal{I}' \models \mathcal{R}'$.

For the converse direction, let $\mathcal{I}'$ be a model of D' w.r.t. $\mathcal{T}'$ and $\mathcal{R}'$. $\mathcal{I}$ will be identical to $\mathcal{I}'$ in every respect except for concrete roles and datatype group-based concepts $C_1, \ldots, C_k$. We can construct $C_i^{\mathcal{I}}$ $(1 \leq i \leq k)$ as $C_i^{\mathcal{I}} = A^{\mathcal{I}'}_{C_i}$ and the interpretations of concrete roles as follows: Let $C = C_1^{\mathcal{I}} \cup \ldots \cup C_k^{\mathcal{I}}$. For each $x_j \in C$, there exists a set $\{C_{j_1}, \ldots, C_{j_{n_x}}\}$ s.t. for each $C_{j_h} \in \{C_{j_1}, \ldots, C_{j_{n_x}}\}, x_j \in C_{j_h}^{\mathcal{I}}$. Let $S_j = C_{j_1} \sqcap \ldots \sqcap C_{j_{n_x}}$. Obviously, $\mathcal{I} \models S_j$. Due to claim (i), the datatype query $\pi(S_j)$ is decidable; therefore, there exists a datatype interpretation $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$ and a solution δ of $\pi(S_j)$ w.r.t. $(\Delta_{\mathrm{D}}, \cdot^{\mathrm{D}})$. Let T be a concrete role, $V_T^{(j)}$ the set of variables in $\pi(S_j)$ that match T, $\delta(V_T^{(j)})$ the set of data values to which δ maps the set of variables in $V_T^{(j)}$. Initially, we set all $T^{\mathcal{I}}$ as $\emptyset$, then for each T used in each S_j, we have $T^{\mathcal{I}} = T^{\mathcal{I}} \cup \{S_j^{\mathcal{I}} \times \delta(V_T^{(j)})\}$. Obviously, we have $\mathcal{I} \models D$. Due to claim (ii) and the construction of $\mathcal{T}'$, we have $\mathcal{I} \models \mathcal{T}$. Due to the definition of match, the constructions of $\mathcal{R}'$ and the interpretations of concrete roles, we have $\mathcal{I} \models \mathcal{R}$. □

Since OWL-Eu corresponds to the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL, which is a sub-language of $\mathcal{SHOIQ}(\mathcal{G}_1)$, we have the following corollary.

Corollary 1. *The OWL-Eu-concept satisfiability problem w.r.t. a knowledge base is decidable.*

Lemma 2. (Tobies[24, Lemma 5.3]) *If $\mathcal{L}$ is a DL that provides the nominal constructor, knowledge base satisfiability can be polynomially reduced to satisfiability of TBoxes and RBoxes.*

According to Corollary 1 and Lemma 5.3, we have the following theorem.

Theorem 2. *The knowledge base satisfiability problem of OWL-Eu is decidable.*

5.4. Overcoming the limitations of OWL datatyping

This section summarises how OWL-Eu overcomes the limitations of OWL datatyping presented in Section 4. Firstly, OWL-Eu is a decidable extension (Theorem 1) of OWL DL that supports customised datatypes with unary datatype expressions (cf. Example 4). Secondly, Definition 12 defines the negations of datatype expressions and OWL-Eu provides relativised negated datatype expression (Definition 11). Thirdly, according to Definition 12, the datatype domain in an interpretation of a datatype group is a superset of (instead of equivalent to) the value spaces of primitive base datatypes and plain literals; hence, typed literals with unsupported predicates are interpreted more intuitively.

6. Related work

The concrete domain approach [2,14] provides a rigorous treatment of datatype predicates, rather than datatypes.[12] In the type system approach [11], datatypes are considered to be sufficiently structured by type systems; however, it does not specify how the derivation mechanism of a type system affects the set of datatypes **D**. An early version of [5] suggests some solutions to the problem of referring to an XML Schema user defined simple type with a URI reference; however, it does not address the computability issue of combining the $\mathcal{SHOIN}$ DL with customised datatypes. The current version of this W3C technical report refers to our work on unary datatype groups, as a solution to the problem of combining OWL DL with customised datatypes. It is worth mentioning that the SPARQL query language for RDF [20] allows the use not only of datatypes, but also of some datatype predicates and operators defined in [15]. SPARQL does not, however, allow the use of customised datatypes or datatype predicates. Furthermore, the eq operator SPARQL supports is not an equivalence relation because of some so-called "corner cases" [5].

7. Conclusion

Although OWL is rather expressive, it has a very serious limitation on datatypes; i.e., it does not support customised datatypes. It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome. Accordingly, the Semantic Web Best Practices and Development Working Group has set up a task force to address this issue. As discussed above, a solution to the problem should cover much more than just a standard way of referring to an XML Schema user defined simple type with a URI reference.

In this paper, we propose OWL-Eu, an extension of OWL DL that supports customised datatypes. The underpinning of OWL-Eu is the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL, a combination of $\mathcal{SHOIN}$ and a unary datatype group. OWL-Eu is decidable if the combined unary datatype group is conforming; conformance of a unary datatype group precisely specifies the conditions on the set of supported datatypes. OWL-Eu provides a general framework for integrating OWL DL with customised datatypes, such as XML Schema non-list simple types.

[12] The reader is referred to Section 5.1.3 of [16] for detailed discussions on concrete domains.

We have implemented a prototype extension of the FaCT [9] DL system, called FaCt-DG, to support TBox reasoning in the $\mathcal{SHIQ}(\mathcal{G}_1)$ DL, a sub-language of OWL-Eu. As for future work, we are planing to extend the DIG1.1 interface [7] to support OWL-Eu, and to implement a protégé [13] plug-in to support XML Schema non-list simple types, i.e. users should be able to define and/or import customised XML Schema non-list simple types based on a set of supported datatypes, and to exploit our prototype through the extended DIG interface.

References

[1] H. Alvestrand, Rfc 3066 – tags for the identification of languages, Technical report, IETF, January 2001, http://www.isi.edu/in-notes/rfc3066.txt.

[2] F. Baader, P. Hanschke, A Schema for integrating concrete domains into concept languages, Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI'91), 1991, pp. 452–457.

[3] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D.L. McGuinness, P.F. Patel-Schneider, L.A. Stein (Eds.), OWL Web Ontology Language Reference, http://www.w3.org/TR/owl-ref/, February 2004.

[4] P.V. Biron, A. Malhotra, Extensible Markup Language (XML) Schema Part 2: Datatypes – W3C Recommendation 02 May 2001, Technical report, World Wide Web Consortium, 2001, http://www.w3.org/TR/xmlschema-2/.

[5] J.J. Carroll, J.Z. Pan, XML Schema Datatypes in RDF and OWL, Technical report, W3C Semantic Web Best Practices and Development Group, November 2004, Editors' Draft, http://www.w3.org/2001/sw/BestPractices/XSCH/xsch-sw/.

[6] Unicode Consortium, The Unicode Standard, Addison-Wesley, 2000, ISBN 0-201-61633-5, version 3.

[7] DIG, SourceForge DIG Interface Project, http://sourceforge.net/projects/dig/, 2004.

[8] P. Hayes, RDF semantics, Technical report, W3C, February 2004, W3C recommendation, http://www.w3.org/TR/rdf-mt/.

[9] I. Horrocks, Using an expressive description logic: fact or fiction? Proceedings of the KR'98, 1998, pp. 636–647.

[10] I. Horrocks, P.F. Patel-Schneider, F. van Harmelen, From SHIQ and RDF to OWL: the making of a web ontology language, J. Web Semantics 1 (1) (2003) 7–26.

[11] I. Horrocks, U. Sattler, Ontology reasoning in the $\mathcal{SHOQ}(D)$ description logic, Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001), 2001, pp. 199–204.

[12] I. Horrocks, U. Sattler, S. Tobies, Practical reasoning for expressive description logics, Proceedings of the International Conference on Logic for Programming and Automated Reasoning (LPAR'99), 1999 number 1705 in LNAI, pp. 161–180.

[13] H. Knublauch, R.W. Fergerson, N.F. Noy, M.A. Musen, The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications, International Semantic Web Conference, 2004, pp. 229–243.

[14] C. Lutz, The complexity of reasoning with concrete domains, Ph.D. thesis, Teaching and Research Area for Theoretical Computer Science, RWTH Aachen, 2001.

[15] A. Malhotra, J. Melton, N. Walsh, XQuery 1.0 and XPath 2.0 Functions and Operators, W3C Working Draft, http://www.w3.org/TR/xpath-functions, July 2004.

[16] J.Z. Pan, Description logics: reasoning support for the semantic web, Ph.D. thesis, School of Computer Science, The University of Manchester, Oxford Road, Manchester M13 9PL, UK, 2004.

[17] J.Z. Pan, I. Horrocks, Extending datatype support in web ontology reasoning, Proceedings of the 2002 International Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002), October 2002.

[18] J.Z. Pan, I. Horrocks, Web ontology reasoning with datatype groups, Proceedings of the 2003 International Semantic Web Conference (ISWC2003), 2003, pp. 47–63.

[19] P.F. Patel-Schneider, P. Hayes, I. Horrocks, OWL web ontology language semantics and abstract syntax, Technical report, W3C, February 2004, W3C Recommendation, http://www.w3.org/TR/2004/REC-owl-semantics-20040210.

[20] E. Prud'hommeaux, A. Seaborne (Eds.), SPARQL Query Language for RDF, W3C Working Draft, http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012, December 2004.

[21] RDF-Logic Mailing List, http://lists.w3.org/archives/public/www-rdf-logic/. W3C Mailing List, starts from 2001.

[22] A. Rector, Re: [UNITS, OEP] FAQ: Constraints on data values range, Discussion in [23], April 2004, http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0216.html.

[23] Semantic Web Best Practice and Development Working Group Mailing List, http://lists.w3.org/archives/public/public-swbp-wg/, W3C Mailing List, starts from 2004.

[24] S. Tobies, Complexity results and practical algorithms for logics in knowledge representation, Ph.D. thesis, Rheinisch-Westf'alischen Technischen Hochschule Aachen, 2001, http://lat.inf.tu-dresden.de/research/phd/Tobies-PhD-2001.pdf.