Introducing Customised Datatypes and Datatype Predicates into OWL[(*)]

Jeff Z. Pan and Ian Horrocks

School of Computer Science, University of Manchester, UK

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 Deployment Working Group has set up a task force to address this issue. This paper provides a solution for this issue by presenting two decidable datatype extensions of OWL DL, namely OWL-Eu and OWL-E. OWL-Eu provides a minimal extension of OWL DL to support customised datatypes, while OWL-E extends OWL DL with both customised datatypes and customised datatype predicates.

1 Introduction

The OWL Web Ontology Language [1] is a W3C recommendation for expressing ontologies in the Semantic Web. Datatype support [7, 8] is one of the key features that OWL is expected to provide, and has prompted extensive discussions in the RDF-Logic mailing list [10] and in the Semantic Web Best Practices mailing list [12]. 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, such as XML Schema derived datatypes.

It has been pointed out that many potential users will not adopt OWL unless this limitation is overcome [11], as it is often necessary to enable users to define their own datatypes and datatype predicates for their ontologies and applications. One of the most well known type systems is W3C XML Schema Part 2 [2], which defines facilities to allow users to define customised datatypes, such as those defined by imposing some restrictions in the value spaces of existing datatypes.

Example 1. Customised datatypes are useful in capturing the intended meaning of some vocabulary in ontologies. For example, users might want to use the customised datatype 'atLeast18' in the following definition of the class 'Adult':

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

which says that an Adult is a Person whose age is at least 18. The datatype constraint

[(*)] This work is partially supported by the FP6 Network of Excellence EU project Knowledge Web (IST-2004-507842).

'at least 18' can be defined as an XML Schema user-defined datatype

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

in which the facet 'minInclusive' is used to restrict the value space of `atLeast18` (a customised datatype) to be a subset of the value space of `integer` (an XML Schema built-in datatype).

User-defined datatypes (like the above one) cannot, however, be used in the OWL datatyping, which (only) provides the use of some *built-in* XML Schema datatypes and enumerated datatypes, which are defined by explicitly specifying their instances. The OWL datatyping does not support XML Schema customised datatypes for the following two reasons: (i) XML Schema does not provide a standard way to access a user-defined datatype. (ii) OWL DL does not provide a mechanism to guarantee the computability of the kinds of customised datatypes it supports.

This paper provides a solution for this issue by presenting two decidable datatype extensions of OWL DL, namely OWL-Eu and OWL-E. OWL-Eu provides a minimal extension of OWL DL to support customised datatypes, while OWL-E extends OWL DL with both customised datatypes and customised datatype predicates. The rest of the paper is organised as follows: Section 2 further discusses the motivations of introducing customised datatypes and datatype predicates. Section 3 extends the OWL datatyping to unary datatype groups, which enables the use of customised datatypes. Section 4 and 5 present the OWL-Eu and the OWL-E languages, respectively; the latter one is based on datatype groups, which are general forms of unary datatype groups. Section 6 concludes the paper and suggests some future work.

2 Motivations

Allowing users to define their own vocabulary is one of the most useful features that ontologies can provide over other approaches, such as the Dublin Core, of providing semantics in the Semantic Web. In the Dublin Core standard, the meaning of the set of 15 information properties are described in English text. The main drawback of the Dublin Core is its inflexibility; it is impossible to 'predefine' information properties for all sorts of applications.

Ontologies, however, are more flexible in that users can define their own vocabulary based on existing vocabularies. In ontology languages, a set of class constructors are usually provided so that users can build class expressions based on, for example, existing class names. The intended meaning of the vocabulary, therefore, can be captured by the axioms in the ontologies. Let us revisit Example 1 and consider the intended meaning of the Adult class. According to its definition, an Adult is a Person who is at least 18 years old. As a result, programs can also understand the meaning of customised vocabulary, with the help of ontologies.

Although OWL DL provides a set of expressive class constructors to build customised classes, it does not provide enough expressive power to support, for example, XML Schema customised datatypes. In order to capture the intended meaning of Adult, Example 1 has already shown the necessity of customised datatypes. In what follows, we give some more examples to illustrate the usefulness of customised datatypes and datatype predicates in various SW and ontology applications.

Example 2. **Semantic Web Service: Matchmaking**
Matchmaking is a process that takes a service requirement and a group of service advertisements as input, and returns all the advertisements that may potentially satisfy the requirement. In a computer sales ontology, a service requirement may ask for a PC with memory size greater than 512Mb, unit price less than 700 pounds and delivery date earlier than 15/03/2004.

Here 'greater than 512', 'less than 700' and 'earlier than 15/03/2004' are customised datatypes of base datatypes integer, integer and date, respectively.

Example 3. **Electronic Commerce: A 'No Shipping Fee' Rule**
Electronic shops may need to classify items according to their sizes, and to reason that an item for which the sum of height, length and width is no greater than 15cm belongs to a class in their ontology, called 'small-items'. Then they can have a rule saying that for 'small-items' no shipping costs are charged. Accordingly, the billing system will charge no shipping fees for all the instances of the 'small-items' class.

Here 'greater than 15' is a customised datatype, 'sum' is a datatype predicate, while 'sum no greater than 15' is a customised datatype predicate.

3 Unary Datatype Groups

The OWL datatyping is defined based on the notion of datatype maps [9]. A datatype map is a partial mapping from supported datatype URIrefs to datatypes. In this section, we introduce unary datatype groups, which extend the OWL datatyping with a hierarchy of supported datatypes.

Definition 1 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 exist unary datatype URI reference $\mathrm{rdfs{:}Literal}, \mathrm{owlx{:}DatatypeBottom} \notin \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$, if $u \in \mathbf{B}, \mathsf{dom}(u) = u$; otherwise, $\mathsf{dom}(u) = v$, where $v \in \mathbf{B}$. $\diamond$

Definition 1 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 4. $\mathcal{G}_1 = (\mathbf{M}_{d1}, \mathbf{B}_1, \mathsf{dom}_1)$ is a unary datatype group, where

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

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

According to $\mathbf{M}_{d1}$, we have $\mathbf{S}_1 = \{\text{xsd:integer}, \text{xsd:string}, \text{xsd:nonNega-}$ tiveInteger, $\text{xsdx:integerLessThanN}\}$, hence $\mathbf{B}_1 \subseteq \mathbf{S}_1$. ♢

Based on a unary datatype group, we can provide a formalism (called datatype expressions) for constructing customised datatypes using supported datatypes.

Definition 2 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 5 on page 8), abbreviated $\mathbf{Dexp}(\mathcal{G})$, is inductively defined as follows:

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

Example 5. $\mathcal{G}$-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 4 (page 3),

– 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 = "expensive"/>
      </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)
).

Note that "low"^^xsd:string is a typed literal, which represents a value of the xsd:string datatype. "low", instead, is a plain literal, where no datatype information is provided. ♢

We now define the interpretation of a unary datatype group.

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

Table 1. Syntax and semantics of datatype expressions (OWL-Eu data ranges)

Definition 3 A *datatype interpretation* $\mathcal{I}_{\mathbf{D}}$ of a unary datatype group $\mathcal{G} = (\mathbf{M}_d, \mathbf{B}, \mathsf{dom})$ is a pair $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$, where $\Delta_{\mathbf{D}}$ (the datatype domain) is a non-empty set and $\cdot^{\mathbf{D}}$ is a datatype interpretation function, which has to satisfy the following conditions:

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

Moreover, we extend $\cdot^{\mathbf{D}}$ to $\mathcal{G}$ unary datatype expression as shown in Table 5 (page 8). Let E be a $\mathcal{G}$ unary datatype expression, the negation of E is of the form $\neg E$, which is interpreted as $\Delta_{\mathbf{D}} \setminus E^{\mathbf{D}}$. $\diamond$

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

Definition 4 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

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

where the v_j are variables from $\mathbf{V}$, $v_1^{(i)}, v_2^{(i)}$ are variables 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 $\mathscr{C}$ is called *satisfiable* iff there exists an interpretation $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$ of $\mathcal{G}$ and a function δ mapping the variables in $\mathscr{C}$ to data values in $\Delta_{\mathbf{D}}$ s.t. $\delta(v_j) \in u_j^{\mathbf{D}}$ (for all $1 \leq j \leq k$) and $\{\delta(v_1^{(i)}), \delta(v_2^{(i)})\} \subseteq u_i^{\mathbf{D}}$ 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 $\mathscr{C}$ w.r.t. $(\Delta_{\mathbf{D}}, \cdot^{\mathbf{D}})$. $\diamond$

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

Definition 5 A unary datatype group $\mathcal{G}$ is *conforming* iff

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

4 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 (i.e., enumerated datatypes as well as some built-in XML Schema datatypes) to OWL-Eu data ranges defined as follows.

Definition 6 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 5 (page 8). $\diamond$

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. Hence, this extension of OWL DL is as large as is necessary to support customised datatypes.

Example 6. 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 5 on page 8):

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

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

It turns out that OWL-Eu (i.e., the $\mathcal{SHOIN}(\mathcal{G}_1)$ DL) is decidable.

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

Proof: (Sketch) We will show the decidability of $\mathcal{SHOIN}(\mathcal{G}_1)$-concept satisfiability w.r.t. TBoxes and RBoxes by reducing it to the $\mathcal{SHOIN}$-concept satisfiability w.r.t. TBoxes and RBoxes. 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 $>_1(v) \wedge \leq_0(v)$).

5 OWL-E: A Step Further

In this section, we present a further extension of OWL-Eu, called OWL-E, which supports not only customised datatypes, but also customised datatype predicates.

A *datatype predicate* (or simply *predicate*) p is characterised by an arity $a(p)$, or a minimum arity $a_{min}(p)$ if p can have multiple arities, and a predicate extension (or simply *extension*) $E(p)$. The notion of predicate maps can be defined in an obvious way. For example, $=^{int}$ is a (binary) predicate with arity $a(=^{int}) = 2$ and extension $E(=^{int}) = \{\langle i_1, i_2\rangle \in V(\texttt{integer})^2 \mid i_1 = i_2\}$, where $V(\texttt{integer})$ is the value space for the datatype `integer`.

Now we can generalise unary datatype groups by the definition of datatype groups. In fact, datatypes and datatype predicates can be unified in datatype groups. Roughly speaking, a datatype group is a group of built-in predicate URIrefs 'wrapped' around a set of primitive datatype URIrefs. A *datatype group* $\mathcal{G}$ is a tuple ($\mathbf{M}_p$,$\mathbf{B}$,dom), where $\mathbf{M}_p$ is the *predicate map* of $\mathcal{G}$, $\mathbf{B}$ is the set of *primitive datatype* URI references in $\mathcal{G}$ and dom is the *declared domain function*. We call $\mathbf{S}$ the set of built-in predicate URI references of $\mathcal{G}$, i.e., for each $u \in \mathbf{S}$, $\mathbf{M}_p(u)$ is defined; we require $\mathbf{B} \subseteq \mathbf{S}$. The declared domain function dom has the following properties: for each $u \in \mathbf{S}$,

$$\mathsf{dom}(u) = \begin{cases} u & \text{if } u \in \mathbf{B}, \\ (v_1, \ldots, v_n), \text{ where } v_1, \ldots, v_n \in \mathbf{B} & \text{if } u \in \mathbf{S} \setminus \mathbf{B} \text{ and} \\ & a(\mathbf{M}_p(u)) = n, \\ \{(\underbrace{v, \ldots, v}_{i \text{ times}}) \mid i \geq n\}, \text{ where } v \in \mathbf{B} & \text{if } u \in \mathbf{S} \setminus \mathbf{B} \text{ and} \\ & a_{min}(\mathbf{M}_p(u)) = n. \end{cases}$$

Example 7. $\mathcal{G}_2 = (\mathbf{M}_{p_2}, \mathbf{B}_2, \mathsf{dom}_2)$ is a datatype group, where

- $\mathbf{M}_{p_2} = \{\text{xsd:integer} \mapsto \texttt{integer}, \text{xsd:string} \mapsto \texttt{string}, \text{xsd:integerGreaterThanOrEqualToN} \mapsto \geq_{\text{N}}, \text{xsdx:integerLessThanN} \mapsto <_{\text{N}}, \text{xsdx:integerEquality} \mapsto =^{\texttt{int}}\}$,
- $\mathbf{B}_2 = \{\text{xsd:string, xsd:integer}\}$, and
- $\mathsf{dom}_2 = \{\text{xsd:integer} \mapsto \text{xsd:integer}, \text{xsd:string} \mapsto \text{xsd:string}, \text{xsd:integerGreaterThanOrEqualToN} \mapsto \text{xsd:integer}, \text{xsdx:integerLessThanN} \mapsto \text{xsd:integer}, \text{xsdx:integerEquality} \mapsto (\text{xsd:integer}, \text{xsd:integer})\}$.

According to $\mathbf{M}_{p_2}$, we have $\mathbf{S}_2 = \{\text{xsd:integer}, \text{xsd:string}, \text{xsd:nonNegativeInteger}, \text{xsdx:integerLessThanN}, \text{xsdx:integerEquality}\}$, hence $\mathbf{B}_2 \subseteq \mathbf{S}_2$. $\diamondsuit$

Furthermore, based on datatype groups, we can extend unary datatype expressions to general (n-ary) datatype expressions. While enumerated expressions remain the same, relativised negated, conjunctive and disjunctive unary datatype expressions can be easily extended to the n-nary case. There is a new kind of datatype expression called *domain expression*: $\texttt{domain}(u_1, \ldots, u_n)$, where u_i is either rdfs:Literal or supported unary datatype predicate URIrefs, or their relativised negations. For example, the customised predicate 'sumNoGreaterThanOrEqualTo15', with extension $E(sumNoGreaterThanOrEqualTo15) = \{\langle i_0, i_1, i_2, i_3\rangle \in V(\texttt{integer})^4 \mid i_0 =$

Abstract Syntax	DL Syntax	Semantics
not(u)	$\overline{u}$	$\Delta_{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{B}$ $(\mathsf{dom}(u))^{\mathbf{D}} \setminus u^{\mathbf{D}}$ if $u \in \mathbf{S} \setminus \mathbf{B}$ $\bigcup_{n \geq 1}(\Delta_{\mathbf{D}})^n \setminus u^{\mathbf{D}}$ otherwise
domain($u_1, \ldots, u_n$)	$[u_1, \ldots, u_n]$	$u_1^{\mathbf{D}} \times \ldots \times u_n^{\mathbf{D}}$

Table 2. Syntax and semantics of (new) datatype expressions

$i_1 + i_2 + i_3$ and $\neg(i_0 \geq 15)\}$ and arity $a(sumNoGreaterThanOrEqualTo15) = 4$, can be represented by

$$\text{xsdx:integerAddition} \wedge [\,\overline{\text{xsdx:integerGreaterThanOrEqualTo15}}, \text{xsd:integer}, \text{xsd:integer}, \text{xsd:integer}\,],$$

which is a conjunctive expression, where the first conjunct is a predicate URIref (that represents $+^{int}$) and the second conjunct is a domain expression.

We can extend the datatype interpretation $\mathcal{I}_{\mathbf{D}}$ presented in Definition 3 to give semantics to datatype groups. For each $u \in \mathbf{S}$, $u^{\mathbf{D}} = E(\mathbf{M}_p(u)) \subseteq (\mathsf{dom}(u))^{\mathbf{D}}$, where $(\mathsf{dom}(u))^{\mathbf{D}}$ is defined as follows: if $\mathsf{dom}(u) = (d_1, \ldots, d_n)$ and $a(\mathbf{M}_p(u)) = n$, then $(\mathsf{dom}(u))^{\mathbf{D}} = d_1^{\mathbf{D}} \times \ldots \times d_n^{\mathbf{D}}$; if $\mathsf{dom}(u) = \{(\underbrace{d, \ldots, d}_{i \text{ times}}) \mid i \geq n\}$ and $a_{min}(\mathbf{M}_p(u)) = n$, then $(\mathsf{dom}(u))^{\mathbf{D}} = \bigcup_{i \geq n} (d^{\mathbf{D}})^i$. The abstract syntax, DL syntax and semantics of relativised negated and domain expressions are presented in Table 5.

The following definition summarises the conditions that computable datatype groups require.

Definition 7 (Conforming Datatype Group) A datatype group $\mathcal{G}$ is *conforming* iff

1. for any $u \in \mathbf{S} \setminus \mathbf{B}$ with $a(\mathbf{M}_p(u)) = n \geq 2$: $\mathsf{dom}(u) = (\underbrace{w, \ldots, w}_{n \text{ times}})$ for some $w \in \mathbf{B}$, and
2. for any $u \in \mathbf{S} \setminus \mathbf{B}$: there exist $u' \in \mathbf{S} \setminus \mathbf{B}$ such that $u'^{\mathbf{D}} = \overline{u}^{\mathbf{D}}$, and
3. the satisfiability problem for finite negation-free predicate conjunctions is decidable, and
4. for each primitive datatype URIref $u_i \in \mathbf{B}$, there exists $w_i \in \mathbf{S}$, s.t. $\mathbf{M}_p(w_i) = \neq_{u_i}$ where $\neq_{u_i}$ is the binary inequality predicate for $\mathbf{M}_p(u_i)$. $\diamond$

Finally, OWL-E extends OWL-Eu with the datatype group-related class constructors presented in Table 3.

Example 8. (OWL-E classes)

Assume that electronic-shops want to define small items as items of which the sum of height, length and width is no greater than or equal to 15cm. The SmallItem class can be represented by the following datatype group-based concept description:

$$\exists T_s, T_h, T_l, T_w.(+^{int} \wedge [\overline{\geq_{15}}, integer, integer, integer]),$$

New Element	DL Syntax	Semantics
expressive predicate exists restriction	$\exists T_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \exists t_1,\ldots,t_n.\langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate value restriction	$\forall T_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \forall t_1,\ldots,t_n.\langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \rightarrow \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\}$
expressive predicate atleast restriction	$\geqslant mT_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1,\ldots,t_n\rangle \mid \langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\} \geq m\}$
expressive predicate atmost restriction	$\leqslant mT_1,\ldots,T_n.E$	$\{x \in \Delta^{\mathcal{I}} \mid \sharp\{\langle t_1,\ldots,t_n\rangle \mid \langle x,t_i\rangle \in T^{\mathcal{I}}$ (for all $1 \leq i \leq m) \wedge \langle t_1,\ldots,t_n\rangle \in E^{\mathbf{D}}\} \leq m\}$

Table 3. New class constructors in OWL-E

where T_s, T_h, T_l, T_w are concrete roles representing "sum in cm", "hight in cm", "length in cm" and "width in cm", respectively, and $(+^{int} \wedge [\overline{\geq_{15}}, integer, integer, integer])$ is a conjunctive datatype expression representing the customised predicate "sum no larger than or equal to 15".[1] $\diamondsuit$

Like OWL-Eu, OWL-E (i.e., the $\mathcal{SHOIQ}(\mathcal{G})$ DL) is also a decidable extension of OWL-DL.

Theorem 2. *The $\mathcal{SHOIN}(\mathcal{G})$- and $\mathcal{SHOIQ}(\mathcal{G})$-concept satisfiability and subsumption problems w.r.t. TBoxes and RBoxes are decidable.*

According to Tobies [13, 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. Hence, we obtain the following theorem.

Theorem 3. *The knowledge base satisfiability problems of $\mathcal{SHOIN}(\mathcal{G})$ and $\mathcal{SHOIQ}(\mathcal{G})$ are decidable.*

6 Conclusion

In this paper, we propose OWL-Eu and OWL-E, two decidable extensions of OWL DL that support customised datatypes and customised datatype predicates. OWL-Eu provides a general framework for integrating OWL DL with customised datatypes, such as XML Schema non-list simple types. OWL-E further extends OWL-Eu to support customised datatype predicates.

We have implemented a prototype extension of the FaCT [5] DL system, called FaCT-DG, to support TBox reasoning in both OWL-Eu and OWL-E (without nominals). As for future work, we are planning to extend the DIG1.1 interface [3] to support OWL-Eu, and to implement a Protégé [6] 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. Furthermore, we plan to extend the FaCT++ DL reasoner [4] to support the full OWL-Eu and OWL-E ontology languages.

[1] To save space, we use predicates instead of predicate URIrefs here.

Bibliography

[1] Sean Bechhofer, Frank van Harmelen, James Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein eds. OWL Web Ontology Language Reference. http://www.w3.org/TR/owl-ref/, Feb 2004.

[2] Paul V. Biron and Ashok 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/.

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

[4] FaCT++. `http://owl.man.ac.uk/factplusplus/`, 2003.

[5] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of KR'98*, pages 636–647, 1998.

[6] Holger Knublauch, Ray W. Fergerson, Natalya Fridman Noy, and Mark A. Musen. The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. In *International Semantic Web Conference*, pages 229–243, 2004.

[7] Jeff Z. Pan and Ian Horrocks. Extending Datatype Support in Web Ontology Reasoning. In *Proc. of the 2002 Int. Conference on Ontologies, Databases and Applications of SEmantics (ODBASE 2002)*, Oct 2002.

[8] Jeff Z. Pan and Ian Horrocks. Web Ontology Reasoning with Datatype Groups. In *Proc. of the 2003 International Semantic Web Conference (ISWC2003)*, pages 47–63, 2003.

[9] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. Technical report, W3C, Feb. 2004. W3C Recommendation, URL `http://www.w3.org/TR/2004/REC-owl-semantics-20040210/`.

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

[11] Alan Rector. Re: [UNITS, OEP] FAQ : Constraints on data values range. Discussion in [12], Apr. 2004. http://lists.w3.org/Archives/Public/public-swbp-wg/2004Apr/0216.html.

[12] 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.

[13] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, Rheinisch-Westfälischen Technischen Hochschule Aachen, 2001. URL `http://lat.inf.tu-dresden.de/research/phd/Tobies-PhD-2001.pdf` .