XXXXX

XXXXX[1] and XXXXX[2]

[1] XXXXX
XXXXX,
[2] XXXXX
XXXXX

Abstract. Keywords: ..

1 Motivation

- mappings from RDF validation requirements (more specifically RDF constraint formulation requirements) to DL
- mappings from RDF validation requirements (more specifically RDF constraint formulation requirements) to generic constraints (expressed by generic constraint language)

Thomas: I think we should provide mappings to DL directly in the requirements chapters and not just in 1 table at the end of the paper

2 Subsumption

requirement:

- R-100-SUBSUMPTION

2.1 Description

- DL terminology: concept inclusion

2.2 Simple Example

DL:
Mother $\sqsubseteq$ Parent

- all mothers are parents
- the concept Mother is subsumed by the concept Parent

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Mother	-	-	Parent	$\sqsubseteq$	

2.3 Simple Example

– Jedis feel the force

```
# DSP
# ---
-
```

```
# OWL2
# ----
:Jedi rdfs:subClassOf :FeelingForce .
```

```
# ReSh
# ----
the extension ext:extendsShape may be used
```

```
# ShEx
# ----
:FeelingForce {
    :feelingForce (true) }
:Jedi {
    & :FeelingForce ,
    :attitute ('good') }
```

```
# data
# ----
:Yoda
    :feelingForce true ;
    :attitute 'good' .
```

DL:
:Jedi $\sqsubseteq$:FeelingForce

2.4 Complex Example

DL:
Rich $\sqsubseteq$ $\neg$ Poor

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	$\neg$ Poor	-	-	Poor	$\neg$	
class	Rich	-	-	$\neg$ Poor	$\sqsubseteq$	

3 Class Equivalence

requirement:

– R-3-EQUIVALENT-CLASSES

3.1 Description

- Concept equivalence asserts that two concepts have the same instances [**?**]
- While synonyms are an obvious example of equivalent concepts, in practice one more often uses concept equivalence to give a name to complex expression [**?**]
- Concept equivalence is indeed subsumption from left and right. ($A \sqsubseteq B$ and $B \sqsubseteq A$ implies $A \equiv B$)

3.2 Simple Example

DL:
Person $\equiv$ Human

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Person	-	-	Human	$\equiv$	
class	Human	-	-	Person	$\equiv$	

maybe better:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Person	-	-	Human	$\sqsubseteq$	
class	Human	-	-	Person	$\sqsubseteq$	

Erman: It is not better since we should stick to what we use as syntax.

4 Sub Object Properties

requirements:

- R-54-SUB-OBJECT-PROPERTIES

4.1 Description

- DL terminology: role inclusion

4.2 Simple Example

DL:
parentOf $\sqsubseteq$ ancestorOf

- states that parentOf is a subrole of ancestorOf , i.e., every pair of individuals related by parentOf is also related by ancestorOf

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	parentOf	ancestorOf	-	-	$\sqsubseteq$	

better:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	parentOf	ancestorOf	$\top$	$\sqsubseteq$	

4.3 Simple Example

DL:
hasDog $\sqsubseteq$ hasPet

5 Sub Data Properties

requirements:

– R-54-SUB-DATA-PROPERTIES

5.1 Description

– DL terminology: concrete domains functional roles

5.2 Simple Example

DL:
$Person \sqcap \forall hasAge.\leqslant_{10} \sqsubseteq Person \sqcap \forall hasEvenAge.\leqslant_{10}$

6 Object Property Paths / Object Property Chains

requirement:

– R-55-OBJECT-PROPERTY-PATHS

6.1 Description

– DL terminology: complex role inclusion axiom / role composition
– role composition can only appear on the left-hand side of complex role inclusions [**?**]

6.2 Simple Example

DL:
brotherOf $\circ$ parentOf $\sqsubseteq$ uncleOf

```
1 # OWL2
2 # ----
3 :uncleOf owl:propertyChainAxiom ( :brotherOf :parentOf ) .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	brotherOf, parentOf	uncleOf	$\top$	$\sqsubseteq$	-

old mapping:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	uncleOf	brotherOf, parentOf	-	$\top$	$\sqsubseteq$	

7 Disjoint Properties

requirements:

– R-9-DISJOINT-PROPERTIES

7.1 Description

–

7.2 Simple Example

DL:
Disjoint(parentOf, childOf)

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	parentOf	childOf	$\top$	$\neq$	

8 Intersection

requirements:

– R-15-CONJUNCTION-OF-CLASS-EXPRESSIONS
– R-16-CONJUNCTION-OF-DATA-RANGES

8.1 Description

– composition / conjunction

DLs allow new concepts and roles to be built using a variety of different constructors. We distinguish concept and role constructors depending on whether concept or role expressions are constructed. In the case of concepts, one can further separate basic Boolean constructors, role restrictions and nominals/enumerations [**?**].

Boolean concept constructors provide basic boolean operations that are closely related to the familiar operations of intersection, union and complement of sets, or to conjunction, disjunction and negation of logical expressions [**?**].

DL:
Mother $\equiv$ Female $\sqcap$ Parent

concept inclusions allow us to state that all mothers are female and that all mothers are parents, but what we really mean is that mothers are exactly the female parents. DLs support such statements by allowing us to form complex concepts such as the intersection (also called conjunction) which denotes the set of individuals that are both female and parents. A complex concept can be used in axioms in exactly the same way as an atomic concept, e.g., in the equivalence Mother $\equiv$ Female $\sqcap$ Parent .

8.2 Simple Example

DL:
Female ⊓ Parent

– complex concept of all individuals which are of the concept Female and of the concept Parent

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Female ⊓ Parent	-	-	Female, Parent	⊓	

DL:
Mother ≡ Female ⊓ Parent

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Mother	-	-	Female, Parent	⊓	

9 Disjunction

requirements:

– R-17-DISJUNCTION-OF-CLASS-EXPRESSIONS
– R-18-DISJUNCTION-OF-DATA-RANGES

9.1 Description

– union / inclusive OR

9.2 Simple Example

DL:
Father ⊔ Mother ⊔ Child

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	complex concept	-	-	Father, Mother, Child	⊔	

9.3 Simple Example

DL:
Parent ≡ Father ⊔ Mother

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Parent	-	-	Father, Mother	⊔	

10 Negation

requirements:

- R-19-NEGATION-OF-CLASS-EXPRESSIONS
- R-20-NEGATION-OF-DATA-RANGES

10.1 Description

- complement

10.2 Simple Example

DL:
$\neg Married$

- set of all individuals that are not married

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	¬ Married	-	-	Married	¬	

10.3 Complex Example

DL:
Female ⊓ ¬ Married

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	¬ Married	-	-	Married	¬	
class	Female ⊓ ¬ Married	-	-	Female, ¬ Married	⊓	

11 Disjoint Classes

requirements:

- R-7-DISJOINT-CLASSES

11.1 Description

-

11.2 Simple Example

DL:
Male ⊓ Female ⊑ ⊥

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Male ⊓ Female	-	-	Male, Female	⊓	
class	⊤	-	-	Male ⊓ Female, ⊥	⊑	

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Male	-	-	Female	≠	
class	Female	-	-	Male	≠	

12 Existential Quantification on Properties

requirements:

- R-86-EXISTENTIAL-QUANTIFICATION-ON-PROPERTIES

12.1 Description

- DL termonology: existential restriction
- An existential class expression ObjectSomeValuesFrom(OPE CE) consists of an object property expression OPE and a class expression CE, and it contains all those individuals that are connected by OPE to an individual that is an instance of CE. Such a class expression can be seen as a syntactic shortcut for the class expression ObjectMinCardinality(1 OPE CE).
- An existential class expression DataSomeValuesFrom(DPE1 ... DPEn DR) consists of n data property expressions DPEi, $1 \leqslant i \leqslant n$, and a data range DR whose arity MUST be n. Such a class expression contains all those individuals that are connected by DPEi to literals lti, $1 \leqslant i \leqslant n$, such that the tuple (lt1 , ..., ltn) is in DR. A class expression of the form DataSomeValuesFrom(DPE DR) can be seen as a syntactic shortcut for the class expression DataMinCardinality(1 DPE DR).

12.2 Simple Example

DL:
∃ parentOf . ⊤

complex concept that describes the set of individuals that are parents of at least one individual (instance of ⊤).

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	∃ parentOf . ⊤	parentOf	-	⊤	∃	

12.3 Simple Example

DL:
∃ parentOf . Female

the complex concept describes those individuals that are parents of at least one female individual, i.e., those that have a daughter.

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	∃ parentOf . Female	parentOf	-	Female	∃	

12.4 Simple Example

DL:
Parent ≡ ∃ parentOf . ⊤

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	Parent	parentOf	-	⊤	∃	

13 Universal Quantification on Properties

requirements:

– R-91-UNIVERSAL-QUANTIFICATION-ON-PROPERTIES

13.1 Description

– DL termonology: universal restriction

Thomas: definition

13.2 Simple Example

DL:
∀ parentOf . Female

– set of individuals all of whose children are female
– also includes those that have no children at all

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	∀ parentOf . Female	parentOf	-	Female	∀	

14 Property Domain

requirements:

- R-25-OBJECT-PROPERTY-DOMAIN
- R-26-DATA-PROPERTY-DOMAIN

14.1 Description

- restrict domain of object and data properties
- DL terminology: domain restrictions on roles
- The language needs to include an easy-to-use vocabulary to declare that a given property is associated with a class, e.g. to populate input forms with appropriate widgets but also constraint checking. In OO terms this is the declaration of a member, field, attribute or association.

Thomas: definition

14.2 Simple Example

DL:
$\exists$ sonOf . $\top \sqsubseteq$ Male

- restrict the domain of sonOf to male individual

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	sonOf	-	Male	domain	

15 Property Range

requirements:

- R-28-OBJECT-PROPERTY-RANGE
- R-35-DATA-PROPERTY-RANGE

15.1 Description

- restrict range of object and data properties
- DL terminology: range restrictions on roles

Thomas: definition

15.2 Simple Example

DL:
$\top \sqsubseteq \forall$ sonOf . Parent

– restrict the range of sonOf to parents

```
# OWL 2
# -----
sonOf rdfs:range Parent .
```

```
# DSP
# ---
:hasDogRange
        a dsp:DescriptionTemplate ;
        dsp:resourceClass owl:Thing ;
        dsp:statementTemplate [
            a dsp:NonLiteralStatementTemplate ;
            dsp:property sonOf ;
            dsp:nonLiteralConstraint [
                a dsp:NonLiteralConstraint ;
                dsp:valueClass Parent ] ] .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	sonOf	-	Parent	range	

16 Class-Specific Property Range

requirements:

– R-29-CLASS-SPECIFIC-RANGE-OF-RDF-OBJECTS
– R-36-CLASS-SPECIFIC-RANGE-OF-RDF-LITERALS

16.1 Description

– restrict range of object and data properties for individuals within specific context
– context: e.g. class
– The values of each member property of a class may be limited by their value type, such as xsd:string or ex:Person.

16.2 Simple Example

– property 'sonOf'
– context: class 'Man'
– range: class 'Parent'
– → individuals of the class 'Man' can only have 'sonOf' relationships to indivuduals of the class 'Parent'

17 Minimum Unqualified Cardinality Restrictions on Properties

requirements:

- R-81-MINIMUM-UNQUALIFIED-CARDINALITY-ON-PROPERTIES
- R-211-CARDINALITY-CONSTRAINTS

description:

- A minimum cardinality expression ObjectMinCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to at least n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing.
- A minimum cardinality expression DataMinCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to at least n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
- unqualified: CE respective DR is NOT stated.

17.1 Simple Example

```
# ShEx
# ----
:Jedi {
    :attitude ('good') }
:JediStudent {
    & :Jedi ,
    :studentOf {}{1} }
:JediMaster {
    & :Jedi ,
    :mentorOf {}{1,2} }
:SuperJediMaster {
    & :Jedi ,
    :mentorOf {}{3,} }
```

- Jedis have the attitude 'good'
- Jedi students are students of exactly 1 resource
- Jedi masters are mentoring at least 1 and at most 2 resources
- Super Jedi masters are mentoring at least 3 resources

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
```

```
    :mentorOf :Anakin .
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':SuperJediMaster' data shape
# ------------------------------------------------------
:Yoda
```

Thomas: ToDO: add DL

18 Minimum Qualified Cardinality Restrictions on Properties

requirements:

- R-75-MINIMUM-QUALIFIED-CARDINALITY-ON-PROPERTIES
- R-211-CARDINALITY-CONSTRAINTS

description:

- A minimum cardinality expression ObjectMinCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to at least n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing.
- A minimum cardinality expression DataMinCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to at least n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
- qualified: CE respective DR is stated.

18.1 Simple Example

DL:
$\geqslant$ 2 childOf . Parent

Thomas: ToDO: check DL

- set of individuals that are children of at least two parents

```
# OWL 2
# -----
owl:Thing
    a owl:Restriction ;
    owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger ;
    owl:onProperty childOf ;
    owl:onClass Parent .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⩾ 2 childOf . Parent	childOf	-	Parent	⩾	2

18.2 Simple Example

DL:
foaf:Person ≡ ⩾ 2 hasName.xsd:string

Thomas: ToDO: check DL

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	foaf:Person	hasName	-	xsd:string	⩾	2

18.3 Simple Example

Thomas: ToDO: OWL 2 DL

```
# ShEx
# ----
:Jedi {
    :attitude ('good') }
:JediStudent {
    & :Jedi ,
    :studentOf @:Jedi{1} }
:JediMaster {
    & :Jedi ,
    :mentorOf @:Jedi{1,2} }
:SuperJediMaster {
    & :Jedi ,
    :mentorOf @:Jedi{3,} }
```

- Jedis have the attitude 'good'
- Jedi students are students of exactly 1 Jedi
- Jedi masters are mentoring at least 1 and at most 2 Jedis
- Super Jedi masters are mentoring at least 3 Jedis

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
    :mentorOf :Anakin .
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':SuperJediMaster' data shape
# ------------------------------------------------------
:Yoda

# Individuals matching the ':JediMaster' data shape
# -------------------------------------------------
:Obi-Wan
```

Thomas: ToDO: add DL

18.4 Complex Example

DL:
Father2Daughters ≡ Man ⊓ (⩾ 2 hasChild.Woman)

Thomas: ToDO: check DL

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⩾ 2 hasChild.Woman	hasChild	-	Woman	⩾	2
class	Father2Daughters	-	-	Man, ⩾ 2 hasChild.Woman	⊓	-

19 Maximum Unqualified Cardinality Restrictions on Properties

requirements:

- R-82-MAXIMUM-UNQUALIFIED-CARDINALITY-ON-PROPERTIES
- R-211-CARDINALITY-CONSTRAINTS

description:

- A maximum cardinality expression ObjectMaxCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to at most n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing.
- A maximum cardinality expression DataMaxCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to at most n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
- unqualified: CE respective DR is NOT stated.

19.1 Simple Example

```
# ShEx
# ----
:Jedi {
    :attitude ('good') }
```

```
:JediStudent {
    & :Jedi ,
    :studentOf {}{1} }
:JediMaster {
    & :Jedi ,
    :mentorOf {}{1,2} }
:SuperJediMaster {
    & :Jedi ,
    :mentorOf {}{3,} }
```

- Jedis have the attitude 'good'
- Jedi students are students of exactly 1 resource
- Jedi masters are mentoring at least 1 and at most 2 resources
- Super Jedi masters are mentoring at least 3 resources

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
    :mentorOf :Anakin .
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':JediMaster' data shape
# -------------------------------------------------
:Obi-Wan
```

Thomas: ToDO: add DL

20 Maximum Qualified Cardinality Restrictions on Properties

requirements:

- R-76-MAXIMUM-QUALIFIED-CARDINALITY-ON-PROPERTIES
- R-211-CARDINALITY-CONSTRAINTS

description:

- A maximum cardinality expression ObjectMaxCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to at most n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing.

– A maximum cardinality expression DataMaxCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to at most n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
– qualified: CE respective DR is stated.

20.1 Simple Example

DL:
$\leqslant$ 2 childOf . Parent

Thomas: ToDO: check DL

– set of individuals that are children of at most two parents

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\leqslant$ 2 childOf . Parent	childOf	-	Parent	$\leqslant$	2

20.2 Simple Example

Thomas: ToDO: OWL 2 DL

```
# ShEx
# ----
:Jedi {
    :attitude ('good') }
:JediStudent {
    & :Jedi ,
    :studentOf @:Jedi{1} }
:JediMaster {
    & :Jedi ,
    :mentorOf @:Jedi{1,2} }
:SuperJediMaster {
    & :Jedi ,
    :mentorOf @:Jedi{3,} }
```

– Jedis have the attitude 'good'
– Jedi students are students of exactly 1 Jedi
– Jedi masters are mentoring at least 1 and at most 2 Jedis
– Super Jedi masters are mentoring at least 3 Jedis

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
    :mentorOf :Anakin .
```

```
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':JediMaster' data shape
# -------------------------------------------------
:Obi-Wan
```

Thomas: ToDO: add DL

21 Exact Unqualified Cardinality Restrictions on Properties

requirements:

- R-80-EXACT-UNQUALIFIED-CARDINALITY-ON-PROPERTIES
- R-211-CARDINALITY-CONSTRAINTS

description:

- An exact cardinality expression ObjectExactCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to exactly n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing. Such an expression is actually equivalent to the expression ObjectIntersectionOf(ObjectMinCardinality(n OPE CE) ObjectMaxCardinality(n OPE CE)).
- An exact cardinality expression DataExactCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to exactly n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
- unqualified: CE respective DR is NOT stated.

21.1 Simple Example

Thomas: ToDo. OWL 2 DL

```
# ShEx
# ----
:JediStudent {
    :studentOf {}{1} }
```

```
# ReSh
# ----
:JediStudent a rs:ResourceShape ;
    rs:property [
        rs:name "studentOf" ;
        rs:propertyDefinition :studentOf ;
        rs:valueShape [ a rs:ResourceShape] ;
        rs:occurs rs:Exactly-one ; ] .
```

– Jedis have the attitude 'good'
– Jedi students are students of exactly 1 resource

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
    :mentorOf :Anakin .
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':JediStudent' data shape
# --------------------------------------------------
:MaceWindu :Obi-Wan :Anakin :Luke
```

Thomas: ToDO: add DL

22 Exact Qualified Cardinality Restrictions on Properties

requirements:

– R-74-EXACT-QUALIFIED-CARDINALITY-ON-PROPERTIES
– R-211-CARDINALITY-CONSTRAINTS

description:

– An exact cardinality expression ObjectExactCardinality(n OPE CE) consists of a nonnegative integer n, an object property expression OPE, and a class expression CE, and it contains all those individuals that are connected by OPE to exactly n different individuals that are instances of CE. If CE is missing, it is taken to be owl:Thing. Such an expression is actually equivalent to the expression ObjectIntersectionOf(ObjectMinCardinality(n OPE CE) ObjectMaxCardinality(n OPE CE)).
– An exact cardinality expression DataExactCardinality(n DPE DR) consists of a nonnegative integer n, a data property expression DPE, and a unary data range DR, and it contains all those individuals that are connected by DPE to exactly n different literals in DR. If DR is not present, it is taken to be rdfs:Literal.
– qualified: CE respective DR is stated.

22.1 Simple Example

Thomas: ToDO: OWL 2 DL

```
# ShEx
# ----
:Jedi {
    :attitude ('good') }
:JediStudent {
    :studentOf @:Jedi{1} }
```

```
# ReSh
# ----
:Jedi a rs:ResourceShape ;
    rs:property [
        rs:name "attitude" ;
        rs:propertyDefinition :attitude ;
        rs:allowedValue "good" ;
        rs:occurs rs:Exactly-one ;
    ] .
:JediStudent a rs:ResourceShape ;
    rs:property [
        rs:name "studentOf" ;
        rs:propertyDefinition :studentOf ;
        rs:valueShape :Jedi ;
        rs:occurs rs:Exactly-one ;
    ] .
```

- Jedis have the attitude 'good'
- Jedi students are students of exactly 1 Jedi

```
# data
# ----
:Yoda
    :attitude 'good' ;
    :mentorOf :MaceWindu , :Obi-Wan , :Luke .
:MaceWindu
    :attitude 'good' ;
    :studentOf :Yoda .
:Obi-Wan
    :attitude 'good' ;
    :studentOf :Yoda ;
    :mentorOf :Anakin .
:Anakin
    :attitude 'good' ;
    :studentOf :Obi-Wan .
:Luke
    :attitude 'good' ;
    :studentOf :Yoda .
```

```
# Individuals matching the ':JediStudent' data shape
# --------------------------------------------------
:MaceWindu :Obi-Wan :Anakin :Luke
```

Thomas: ToDO: add DL

22.2 Complex Example

DL:
Person $\sqsubseteq \geqslant 2$ childOf . Parent $\sqcap \leqslant 2$ childOf . Parent

Thomas: ToDO: check DL

– every person is a child of exactly two parents

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\geqslant$ 2 childOf . Parent	childOf	-	Parent	$\geqslant$	2
property	$\leqslant$ 2 childOf . Parent	childOf	-	Parent	$\leqslant$	2
class	Person	-	-	$\geqslant$ 2 childOf . Parent, $\leqslant$ 2 childOf . Parent	$\sqcap$	-

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	Person	childOf	-	Parent	$=$	2

23 Inverse Object Properties

requirement:

– R-56-INVERSE-OBJECT-PROPERTIES

23.1 Description

– In many cases properties are used bi-directionally and then accessed in the inverse direction, e.g. parent $\equiv$ child$^-$. There should be a way to declare value type, cardinality etc of those inverse relations without having to declare a new property URI.
– The object property expression OPE1 is an inverse of the object property expression OPE2. Thus, if an individual x is connected by OPE1 to an individual y, then y is also connected by OPE2 to x, and vice versa.

23.2 Simple Example

DL:
parentOf $\equiv childOf^-$

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	parentOf	childOf	-	$\top$	$inverserole(^-)$	

consistent with ontology:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	parentOf	childOf	$\top$	$inverserole(^-)$	

24 Transitive Object Properties

requirements:

– R-63-TRANSITIVE-OBJECT-PROPERTIES

24.1 Description

- Transitivity is a special form of complex role inclusion
- An object property transitivity axiom TransitiveObjectProperty(OPE) states that the object property expression OPE is transitive — that is, if an individual x is connected by OPE to an individual y that is connected by OPE to an individual z, then x is also connected by OPE to z.

24.2 Simple Example

DL:
ancestorOf ∘ ancestorOf ⊑ ancestorOf

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	ancesterOf	ancestorOf, ancestorOf	-	⊤	transitive role	

syntactic sugar (ontology conform):

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⊤	ancestorOf, ancestorOf	ancestorOf	⊤	transitive role	

without syntactic sugar (ontology conform):

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⊤	ancestorOf, ancestorOf	ancestorOf	⊤	⊑	

25 Symmetric Object Properties

requirement:

- R-61-SYMMETRIC-OBJECT-PROPERTIES

25.1 Description

- A role is symmetric if it is equivalent to its own inverse [**?**]
- An object property symmetry axiom SymmetricObjectProperty(OPE) states that the object property expression OPE is symmetric — that is, if an individual x is connected by OPE to an individual y, then y is also connected by OPE to x.

25.2 Simple Example

DL:
marriedTo ≡ $married To^{-}$

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	marriedTo	marriedTo	-	$\top$	$inverserole(^-)$	

ontology conform:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	marriedTo	marriedTo	$\top$	$inverserole(^-)$	

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	marriedTo	-	$\top$	symmetric role	

25.2.1 Asymmetric Object Properties requirement:

– R-62-ASYMMETRIC-OBJECT-PROPERTIES

25.3 Description

– A role is asymmetric if it is disjoint from its own inverse [**?**]
– An object property asymmetry axiom AsymmetricObjectProperty(OPE) states that the object property expression OPE is asymmetric — that is, if an individual x is connected by OPE to an individual y, then y cannot be connected by OPE to x.

25.4 Simple Example

DL:
Disjoint(parentOf , $parentOf^-$)

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$parentOf^-$	parentOf	-	$\top$	inverse role ($^-$)	
property	parentOf	parentOf, $parentOf^-$	-	$\top$	disjoint role ($\neq$)	

ontology conform:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	parentOf	-	$\top$	inverse role ($^-$)	
property	$\top$	parentOf, $parentOf^-$	-	$\top$	disjoint role ($\neq$)	

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	parentOf	-	$\top$	asymmetric role	

25.5 Simple Example

– child parent relations (dbo:child) cannot be symmetric

25.6 Simple Example

– person birth place relations (dbo:birthPlace) cannot be symmetric

26 Class-Specific Reflexive Object Properties / Local Reflexivity

requirement:

–

26.1 Description

– DL terminology: local reflexivity
– set of individuals that are related to themselves via a given role [**?**].
– set of individuals (of a specific class) that are related to themselves via a given object property

26.2 Simple Example

DL:
∃ talksTo .Self.

– set of individuals that are talking to themselves

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	∃ talksTo .Self.	talksTo	-	Self	∃	

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	∃ talksTo .Self.	talksTo	-	-	reflexive role	

27 Reflexive Object Properties / Global Reflexivity

requirement:

– R-59-REFLEXIVE-OBJECT-PROPERTIES

27.1 Description

– DL terminology: reflexive roles, global reflexivity
– global reflexivity can be expressed by imposing local reflexivity on the top concept [**?**]

27.2 Simple Example

DL:
$\top \sqsubseteq \exists$ knows . Self

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	knows	-	$\top$	reflexive role	

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	knows	-	Self	$\exists$	

28 Irreflexive Object Properties

requirement:

- R-60-IRREFLEXIVE-OBJECT-PROPERTIES

28.1 Description

- DL terminology: irreflexive roles
- A role is irreflexive if it is never locally reflexive [**?**]
- An object property irreflexivity axiom IrreflexiveObjectProperty(OPE) states that the object property expression OPE is irreflexive — that is, no individual is connected by OPE to itself.

28.2 Simple Example

DL:
$\top \sqsubseteq \neg \exists$ marriedTo . Self

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	marriedTo	-	$\top$	irreflexive role	

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\exists$ marriedTo . Self	marriedTo	-	Self	$\exists$	
class	$\top$	-	-	$\exists$ marriedTo . Self	$\neg$	

28.3 Simple Example

- a resource cannot be its own parent (dbo:parent)

28.4 Simple Example

- a resource cannot be its own child (dbo:child)

29 Context-Specific Property Groups

requirement:

- R-66-PROPERTY-GROUPS

29.1 Description

29.2 Simple Example

DSCL (ShEx):

```
<Human> {
    (
        foaf:name xsd:string ,
        foaf:givenName xsd:string
    ) ,
    (
        foaf:mbox IRI ,
        foaf:homepage foaf:Document
    ) }
```

- 1. group: 1 foaf:name (range: xsd:string) and 1 foaf:givenName (range: xsd:string) and (,)
- 2. group: 1 foaf:mbox (range: IRI) and 1 foaf:homepage (range: foaf:Document)

DL:
$A \equiv \forall foaf : name.xsd : string$
$B \equiv \forall foaf : givenName.xsd : string$
$C \equiv AXORB$
$D \equiv \forall foaf : mbox.xsd : string$
$Human \equiv C \sqcap D$

30 Context-Specific Exclusive OR of Properties

requirements:

- R-11-CONTEXT-SPECIFIC-EXCLUSIVE-OR-OF-PROPERTIES

30.1 Description

- exclusive or: logical operation that outputs true whenever both inputs differ (one is true, the other is false).
- in OWL 2, inclusive OR of properties would be possible to express, but not exclusive OR of data properties

DL (general):

DL:
$C \sqsubseteq (\neg A \sqcap B) \sqcup (A \sqcap \neg B)$

DL (general) (alternative):

DL:
$C \sqsubseteq (A \sqcup B) \sqcap \neg (A \sqcap B)$

30.2 Simple Example

DSCL (ShEx):

```
<Human> { (
    foaf:name xsd:string |
    foaf:givenName xsd:string ) }
```

DSCL (OWL 2):

```
:Human owl:disjointUnionOf ( :CC1 :CC2 ) .

:CC1 rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty foaf:name ;
    owl:allValuesFrom xsd:string ] .
:CC2 rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty foaf:givenNme ;
    owl:allValuesFrom xsd:string ] .
```

– 1 foaf:name (range xsd:string) XOR 1 foaf:givenName (range xsd:string)

DL:
$A \equiv \forall$ foaf:name . xsd:string
$B \equiv \forall$ foaf:givenName . xsd:string
Human $\sqsubseteq (\neg A \sqcap B) \sqcup (A \sqcap \neg B)$

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	A	foaf:name	-	xsd:string	$\forall$	
property	B	foaf:givenName	-	xsd:string	$\forall$	
class	$\neg$ A	-	-	A	$\neg$	
class	$\neg$ B	-	-	B	$\neg$	
class	$\neg$ A $\sqcap$ B	-	-	$\neg$ A, B	$\sqcap$	
class	A $\sqcap$ $\neg$ B	-	-	A, $\neg$ B	$\sqcap$	
class	Human	-	-	$\neg$ A $\sqcap$ B, A $\sqcap$ $\neg$ B	$\sqcup$	

syntactic sugar:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	A	foaf:name	-	xsd:string	$\forall$	
property	B	foaf:givenName	-	xsd:string	$\forall$	
class	Human	-	-	A, B	XOR	

30.3 Complex Example

DSCL (ShEx):

```
<Human> { (
    foaf:name xsd:string | foaf:givenName xsd:string+ ,
    foaf:familyName xsd:string ) }
```

– 1 foaf:name (range xsd:string) XOR 1-n foaf:givenName (range xsd:string)
– and
– 1 foaf:familyName (range xsd:string)

Thomas: ToDo: DL

Individuals matching the 'Human' data shape:

```
:Han
    foaf:name "Han Solo" ;
    foaf:familyName "Solo" .
:Anakin
    foaf:givenName "Anakin" ;
    foaf:givenName "Darth" ;
    foaf:familyName "Skywalker" .
```

Individual not matching the 'Human' data shape:

```
:Anakin
    foaf:name "Anakin Skywalker" ;
    foaf:givenName "Anakin" ;
    foaf:familyName "Skywalker" .
```

31 Context-Specific Inclusive OR of Properties

requirements:

- R-202-CONTEXT-SPECIFIC-INCLUSIVE-OR-OF-PROPERTIES

31.1 Description

- inclusive or: A logical connective joining two or more predicates that yields the logical value "true" when at least one of the predicates is true.
- context can be a class → constraint applies for individuals of this specific class

31.2 Simple Example

- 1 foaf:name (range: xsd:string) OR 1 foaf:givenName (range: xsd:string)
- it is also possible that both 1 foaf:name and 1 foaf:givenName are stated
- context: class Human

32 Context-Specific Exclusive OR of Property Groups

requirement:

- R-13-DISJOINT-GROUP-OF-PROPERTIES-CLASS-SPECIFIC

32.1 Description

- exclusive or: logical operation that outputs true whenever both inputs differ (one is true, the other is false).

32.2 Simple Example

```
# DSCL (ShEx)
# -----------
<Human> {
    (
        foaf:name xsd:string ,
        foaf:givenName xsd:string )
    |
    (
        foaf:mbox IRI ,
        foaf:homepage foaf:Document ) }
```

- 1. group XOR 2. group
- 1. group: 1 foaf:name (range: xsd:string) and 1 foaf:givenName (range: xsd:string)
- 2. group: 1 foaf:mbox (range: IRI) and 1 foaf:homepage (range: foaf:Document)
- context: class Human

```
# valid data
# ----------
:Thomas
    a :Human ;
    foaf:mbox <thomas.bosch@gesis.org> ;
    foaf:homepage <http://purl.org/net/thomasbosch> .
```

```
# invalid data
# ----------
:Thomas
    a :Human ;
    foaf:name 'Thomas Bosch' ;
    foaf:givenName 'Thomas' ;
    foaf:mbox <thomas.bosch@gesis.org> ;
    foaf:homepage <http://purl.org/net/thomasbosch> .
```

DL:
Human $\sqsubseteq$ ($\neg$ E $\sqcap$ F) $\sqcup$ (E $\sqcap$ $\neg$ F)
E $\equiv$ A $\sqcap$ B
F $\equiv$ C $\sqcap$ D
A $\equiv$ $\forall$ foaf:name . xsd:string
B $\equiv$ $\forall$ foaf:givenName . xsd:string
C $\equiv$ $\forall$ foaf:mbox . IRI
D $\equiv$ $\forall$ foaf:homepage . foaf:Document

32.3 Complex Example

```
# ShExC
# -----
[[
  <UserShape> {                          # A User has
    (foaf:name xsd:string                #  either a foaf:name
     | foaf:givenName xsd:string+,       #   or 1+ givenNames
       foaf:familyName xsd:string),      #   and a family name
    foaf:mbox IRI                        # and an mbox.
  }
]]
```

```
# ReSh
# ----
```

```
[[
  PREFIX rs:<http://open-services.net/ns/core#>
  PREFIX se:<http://www.w3.org/2013/ShEx/Definition#>

  <UserShape> a rs:ResourceShape ;       # A User has
    se:choice [
      rs:property [                      #  either a foaf:name
        rs:name "name" ;
        rs:propertyDefinition foaf:name ;
        rs:valueType xsd:string ;
        rs:occurs rs:Exactly-one ;
      ] ;
      se:ruleGroup [
        rs:property [                    #   or 1+ givenNames
          rs:name "givenName" ;
          rs:propertyDefinition foaf:givenName ;
          rs:valueType xsd:string ;
          rs:occurs rs:One-or-many ;
        ] ;
        rs:property [                    #   and a family name
          rs:name "familyName" ;
          rs:propertyDefinition foaf:familyName ;
          rs:valueType xsd:string ;
          rs:occurs rs:Exactly-one ;
        ] ;
      ] ;
    ] ;
    rs:property [                        # and an mbox.
      rs:name "mbox" ;
      rs:propertyDefinition foaf:mbox ;
      rs:valueType rs:Resource ;
      rs:occurs rs:Exactly-one ;
    ] .
]]
```

33 Context-Specific Inclusive OR of Property Groups

requirements:

–

33.1 Description

– at least one property group must match for individuals of a specific context
– context: may be a class

33.2 Simple Example

– 1. group OR 2. group
– 1. group: 1 foaf:firstName (range: xsd:string) and 1 foaf:lastName (range: xsd:string)
– 2. group: 1 foaf:givenName (range: xsd:string) and 1 foaf:familyName (range: xsd:string)
– context: class Person

```
# valid data
# ----------
:Anakin
    a :Person ;
    foaf:firstName 'Anakin' ;
    foaf:lastName 'Skywalker' ;
    foaf:givenName 'Anakin' ;
    foaf:familyName 'Skywalker' .
```

```
# invalid data
# ----------
:Anakin
    a :Person .
```

34 Allowed Values

requirements:

- R-30-ALLOWED-VALUES-FOR-RDF-OBJECTS
- R-37-ALLOWED-VALUES-FOR-RDF-LITERALS

34.1 Description

- It is a common requirement to narrow down the value space of a property by an exhaustive enumeration of the valid values (both literals or resource). This is often rendered in drop down boxes or radio buttons in user interfaces.
- define a concept by simply enumerating its instances
- A DL nominal is a concept that has exactly one instance
- DL enumeration: combining nominals with union

allowed values for properties

- must be an IRI
- must be an IRI matching this pattern (e.g. http://id.loc.gov/authorities/names/)
- must be an IRI matching one of these patterns
- must be a (any) literal
- must be one of these literals ("red" "blue" "green")
- must be a typed literal of this type (e.g. XML dataType)

34.2 Example

```
# DSP
# ----------
:descriptionTemplate
    a dsp:DescriptionTemplate ;
    dsp:minOccur "0"^^xsd:nonNegativeInteger ;
    dsp:maxOccur "infinity"^^xsd:string ;
    dsp:resourceClass swrc:Book ;
    dsp:statementTemplate [
        a dsp:NonLiteralStatementTemplate ;
        dsp:minOccur "0"^^xsd:nonNegativeInteger ;
```

```
        dsp:maxOccur "infinity"^^xsd:string ;
        dsp:property dcterms:subject ;
        dsp:nonLiteralConstraint [
            a dsp:NonLiteralConstraint ;
            dsp:valueClass skos:Concept ;
            dsp:valueURI :ComputerScience, :SocialScience, :Librarianship ] ] .
```

```
# OWL2
# ----
dcterms:subject rdfs:range :ObjectOneOf .
# EquivalentClasses( :ObjectOneOf ObjectOneOf( :ComputerScience :SocialScience :Librarianship ) )
:ObjectOneOf owl:equivalentClass [
    a owl:Class ;
    owl:oneOf ( :ComputerScience :SocialScience :Librarianship ) ] .
```

– the range of the object property dcterms:subject must consist of the individuals :ComputerScience :SocialScience :Librarianship which are of the class skos:Concept

DL:
$\top \sqsubseteq \forall$ dcterms:subject . skos:Concept $\sqcap$ ($\{ComputerScience\} \sqcup$ {SocialScience} $\sqcup$ {Librarianship})

34.3 Simple Example

DL:
Beatle $\equiv$ {john} $\sqcup$ {paul} $\sqcup$ {george} $\sqcup$ {ringo}

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	Beatle	-	-	$\{john\}$, $\{paul\}$, $\{george\}$, $\{ringo\}$	$\sqcup$	

34.4 Simple Example

DL:
{ComputerScience} $\sqcup$ {SocialScience} $\sqcup$ {Librarianship}

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	complex concept	-	-	{ComputerScience}, {SocialScience}, {Librarianship}	$\sqcup$	-

34.5 Simple Example

– Jedis have blue, green, or white laser swords

```
# DSP
# ---
:personDescriptionTemplate
    a dsp:DescriptionTemplate ;
    dsp:resourceClass :Jedi ;
    dsp:statementTemplate [
        a dsp:LiteralStatementTemplate ;
        dsp:property :laserSwordColor ;
```

```
        dsp:literalConstraint [
            a dsp:LiteralConstraint ;
            dsp:literal "blue" ;
            dsp:literal "green" ;
            dsp:literal "white"] ] .
```

```
# OWL2
# ----
:laserSwordColor rdfs:range :laserSwordColors .
:laserSwordColors
    a rdfs:Datatype .
    owl:oneOf ( "blue" "green" "white" ) .
```

```
# ReSh
# ----
:Jedi a rs:ResourceShape ;
    rs:property [
        rs:name "laserSwordColor" ;
        rs:propertyDefinition :laserSwordColor ;
        rs:allowedValue "blue" , "green" , "white" ;
        rs:occurs rs:Exactly-one ;
    ] .
```

```
# ShEx
# ----
:Jedi {
    :laserSwordColor ('blue' 'green' 'white') }
```

```
# Jedi individuals
# ----------------
:Yoda
    :laserSwordColor 'blue' .
```

DL:
Jedi $\equiv \exists$ laserSwordColor . {'blue'} $\sqcup$ {'green'} $\sqcup$ {'white'}

Thomas: ToDo: check DL

35 Not Allowed Values

requirements:

- R-33-NEGATIVE-OBJECT-CONSTRAINTS
- R-200-NEGATIVE-LITERAL-CONSTRAINTS

35.1 Description

- not allowed objects
- not allowed literals
- definition: A matching triple has any literal / object except those explicitly excluded

35.2 Simple Example

– Siths do not have blue, green, or white laser swords

```
# DSP
# ---
-
```

```
# OWL2
# ----
:laserSwordColor rdfs:range :negativeLaserSwordColors .
:NegativeLaserSwordColors
    a rdfs:Datatype .
    owl:complementOf :laserSwordColors .
:laserSwordColors
    a rdfs:Datatype .
    owl:oneOf ( "blue" "green" "white" ) .
```

```
# ReSh
# ----
-
```

```
# ShEx
# ----
:Sith {
    ! :laserSwordColor ('blue' 'green' 'white') }
```

```
# Sith individuals
# ----------------
:DarthSidious
    :laserSwordColor 'red' .
```

DL:
Sith ≡ ¬(∃ laserSwordColor . {'blue'} ⊔ {'green'} ⊔ {'white'})

Thomas: ToDo: check DL

36 Membership in Controlled Vocabularies

requirements:

– R-32-MEMBERSHIP-OF-RDF-OBJECTS-IN-CONTROLLED-VOCABULARIES
– R-39-MEMBERSHIP-OF-RDF-LITERALS-IN-CONTROLLED-VOCABULARIES

36.1 Description

– resources can only be members of listed controlled vocabularies

36.2 Simple Example

```
# DSCL (DSP)
# ----------
:bookDescriptionTemplate
    a dsp:DescriptionTemplate ;
    dsp:resourceClass swrc:Book ;
    dsp:statementTemplate [
        a dsp:NonLiteralStatementTemplate ;
        dsp:property dcterms:subject ;
        dsp:nonLiteralConstraint [
            a dsp:NonLiteralConstraint ;
            dsp:valueClass skos:Concept ;
            dsp:vocabularyEncodingScheme :BookSubjects, :BookTopics, :BookCategories ] ] .
```

– `skos:Concept` resources can only be members of the listed controlled vocabularies

DL:
$A \equiv skos:Concept \sqcap B$
$B \equiv \forall skos:inScheme.C$
$C \equiv skos:ConceptScheme \sqcap (\{:BookSubjects\} \sqcup \{:BookTopics\} \sqcup \{:BookCategories\})$

```
# valid data
# ----------
:ArtficialIntelligence
    a swrc:Book ;
    dcterms:subject :ComputerScience .
:ComputerScience
    a skos:Concept ;
    dcam:memberOf :BookSubjects ;
    skos:inScheme :BookSubjects .
:BookSubjects
    a skos:ConceptScheme .
```

```
# invalid data
# ------------
:ArtficialIntelligence
    a swrc:Book ;
    dcterms:subject :ComputerScience .
:ComputerScience
    a skos:Concept ;
    dcam:memberOf :BooksAboutBirds ;
    skos:inScheme :BooksAboutBirds ;
    dcam:memberOf :BookSubjects ;
    skos:inScheme :BookSubjects .
:BookSubjects
    a skos:ConceptScheme .
```

– The related subject (`:ComputerScience`) is a member of a controlled vocabulary (`:BooksAboutBirds`) which is not part of the list of allowed controlled vocabularies.

37 IRI Pattern Matching

requirements:

– R-21-IRI-PATTERN-MATCHING-ON-RDF-SUBJECTS
– R-22-IRI-PATTERN-MATCHING-ON-RDF-OBJECTS
– R-23-IRI-PATTERN-MATCHING-ON-RDF-PROPERTIES

37.1 Description

– IRI pattern matching on subjects, properties, and objects

38 Literal Pattern Matching

requirements:

– R-44-PATTERN-MATCHING-ON-RDF-LITERALS

38.1 Description

– pattern matching on literals

38.2 Simple Example

```
# OWL 2 QL (functional-style syntax)
# ----------------------------------
Declaration( Datatype( :SSN ) )
DatatypeDefinition(
    :SSN
    DatatypeRestriction( xsd:string xsd:pattern "[0-9]{3}-[0-9]{2}-[0-9]{4}" ) )
DataPropertyRange( :hasSSN :SSN )
```

```
# OWL 2 QL (turtle syntax)
# ------------------------
:SSN
    a rdfs:Datatype ;
    owl:equivalentClass [
        a rdfs:Datatype ;
        owl:onDatatype xsd:string ;
        owl:withRestrictions (
            [ xsd:pattern "[0-9]{3}-[0-9]{2}-[0-9]{4}" ] ) ] .
:hasSSN rdfs:range :SSN .
```

– A social security number is a string that matches the given regular expression. The second axiom defines :SSN as an abbreviation for a datatype restriction on xsd:string. The first axiom explicitly declares :SSN to be a datatype. The datatype :SSN can be used just like any other datatype; for example, it is used in the third axiom to define the range of the :hasSSN property.

```
# valid data
# ----------
:TimBernersLee
    :hasSSN "123-45-6789"^^:SSN .
```

```
# invalid data
# ------------
:TimBernersLee
    :hasSSN "123456789"^^:SSN .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	SSN	-	-	xsd:string	xsd:pattern	'[0-9]3-[0-9]2-[0-9]4'

39 Negative Literal Pattern Matching

requirements:

– R-44-PATTERN-MATCHING-ON-RDF-LITERALS

39.1 Description

– negative pattern matching on literals

```
# examples
# --------
1. dbo:isbn format is different '!' from '^([iIsSbBnN 0-9-])*$'
2. dbo:postCode format is different '!' from '^[0-9]{5}$'
3. foaf:phone contains any letters ('[A-Za-z]')
```

39.2 Example

```
# test binding (DQTP)
# -------------------
dbo:isbn format is different '!' from '^([iIsSbBnN 0-9-])*$'

P1 => dbo:isbn
NOP => !
REGEX => '^([iIsSbBnN 0-9-])*$'
```

```
# DQTP
# ----
SELECT DISTINCT ?s WHERE { ?s %%P1%% ?value .
    FILTER ( %%NOP%% regex(str(?value), %%REGEX%%) ) }
```

– MATCH Pattern [**?**]
– P1 is the property we need to check against REGEX and NOP can be a not operator (!) or empty.

```
# valid data
# ----------
:FoundationsOfSWTechnologies
    dbo:isbn 'ISBN-13 978-1420090505' .
```

```
# invalid data
# ------------
:HandbookOfSWTechnologies
    dbo:isbn 'DOI 10.1007/978-3-540-92913-0' .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	$\top$	dbo:isbn	-	$\neg$ A	domain	-
class	$\neg$ A	-	-	A	$\neg$	-
class	A	-	-	xsd:string	regex	'^([iIsSbBnN 0-9-])*'

40 Literal Value Comparison

requirement:

– R-43-LITERAL-VALUE-COMPARISON

40.1 Description

examples:

```
# examples
# --------
1. dbo:deathDate before ‘<’ dbo:birthDate
2. dbo:releaseDate after ‘>’ dbo:latestReleaseDate
3. dbo:demolitionDate before ‘<’ dbo:buildingStartDate
```

40.2 Simple Example

```
# DSCL (DQTP)
# -----------
SELECT ?s WHERE {
    ?s %%P1%% ?v1 .
    ?s %%P2%% ?v2 .
    FILTER ( ?v1 %%OP%% ?v2 ) }
```

COMP Pattern [**?**]

Depending on the property semantics, there are cases where two different literal values must have a specific ordering with respect to an operator. P1 and P2 are the datatype properties we need to compare and OP is the comparison operator (<, <=, >, >=, =, !=).

```
# test binding (DQTP)
# -------------------
dbo:deathDate before ‘<’ dbo:birthDate

P1 => dbo:deathDate
P2 => dbo:birthDate
OP => <
```

```
# valid data
# ----------
:AlbertEinstein
    dbo:birthDate ’1879-03-14’^^xsd:date ;
    dbo:deathDate ’1955-04-18’^^xsd:date .
```

```
# invalid data
# ------------
:NeilArmstrong
    dbo:birthDate ’2012-08-25’^^xsd:date ;
    dbo:deathDate ’1930-08-05’^^xsd:date .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⊤	dbo:birthDate	dbo:deathDate	xsd:date	>	-

41 Negative Literal Ranges

requirements:

– R-142-NEGATIVE-RANGES-OF-RDF-LITERAL-VALUES

41.1 Description

- The literal value of a resource (having a certain type) must (not) be within a specific range
- P1 is a data property of an instance of class T1 and its literal value must be between the range of [Vmin,Vmax] or outside ('!').

41.2 Simple Example

- dbo:height of a dbo:Person is not within [0.4,2.5]

41.3 Simple Example

- geo:lat of a spatial:Feature is not within [-90,90]

41.4 Simple Example

- geo:long of a gml:Feature must be in range [-180,180]

42 Literal Ranges

requirement:

- R-45-RANGES-OF-RDF-LITERAL-VALUES

42.1 Description

42.2 Example

```
# OWL 2 DL (functional-style syntax)
# ----------------------------------
Declaration( Datatype( :NumberPlayersPerWorldCupTeam ) )
DatatypeDefinition(
    :NumberPlayersPerWorldCupTeam
    DatatypeRestriction(
        xsd:nonNegativeInteger
        xsd:minInclusive "1"^^xsd:nonNegativeInteger
        xsd:maxInclusive "23"^^xsd:nonNegativeInteger ) )
DataPropertyRange( :position :NumberPlayersPerWorldCupTeam )
```

```
# OWL 2 DL (turtle syntax)
# ------------------------
:NumberPlayersPerWorldCupTeam
    a rdfs:Datatype ;
    owl:equivalentClass [
        a rdfs:Datatype ;
        owl:onDatatype xsd:nonNegativeInteger ;
        owl:withRestrictions (
            [ xsd:minInclusive "1"^^xsd:nonNegativeInteger ]
            [ xsd:maxInclusive "23"^^xsd:nonNegativeInteger ] ) ] .
:position rdfs:range :NumberPlayersPerWorldCupTeam .
```

– The data range 'NumberPlayersPerWorldCupTeam' contains the non negative integers 1 to 23, as each world cup team can only have 23 football players at most

```
# valid data
# ----------
:MarioGoetze
    :position "19"^^:NumberPlayersPerWorldCupTeam .
```

```
# invalid data
# ------------
:MarioGoetze
    :position "99"^^:NumberPlayersPerWorldCupTeam .
```

– range of position is the datatype 1-23
– constraints are stated on this datatype

DL:
1-23 ≡ {'1'} ⊔ {'2'} ⊔ ... ⊔ {'23'}
⊤ ⊑ ∀ position . 1-23

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	AtLeast1	-	-	xsd:nonNegativeInteger	⩾	1
class	AtMost23	-	-	xsd:nonNegativeInteger	⩽	23
class	NumberPlayers...	-	-	AtLeast1, AtMost23	⊓	-
property	⊤	position	-	NumberPlayers...	range	-

or:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
property	⊤	position	-	1-23	range	-
class	1-23	-	-	xsd:nonNegativeInteger	⩾	1
class	1-23	-	-	xsd:nonNegativeInteger	⩽	23

or:

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	1-23	-	-	{'1'}, {'2'}, ..., {'23'}	⊔	-
property	⊤	position	-	1-23	range	-

43 Define Order

requirements:

– R-121-SPECIFY-ORDER-OF-RDF-RESOURCES
– R-217-DEFINE-ORDER-FOR-FORMS/DISPLAY

43.1 Description

– order properties
– order objects of object properties
– order literal of data properties

43.2 Simple Example

44 Validation Levels

requirements:

- R-205-VARYING-LEVELS-OF-ERROR
- R-135-CONSTRAINT-LEVELS
- R-158-SEVERITY-LEVELS-OF-CONSTRAINT-VIOLATIONS
- R-193-MULTIPLE-CONSTRAINT-VALIDATION-EXECUTION-LEVELS

44.1 Description

- Different levels of severity, priority
- validation levels: informational, warning, error, fail, should, recommended, must, may, optional, closed (only this) constraints, open (at least this) constraints

44.2 Simple Example

45 String Operations

requirement:

- R-194-PROVIDE-STRING-FUNCTIONS-FOR-RDF-LITERALS

45.1 Description

- Some constraints require building new strings out of other strings

  examples:

- string length

45.2 Simple Example

46 Context-Specific Valid Classes

requirements:

- R-209-VALID-CLASSES

46.1 Description

- What types of resources (rdf:type) are valid in this context? Context can be an input stream, a data creation function, or an API.

46.2 Simple Example

47 Context-Specific Valid Properties

requirements:

- R-210-VALID-PROPERTIES
- What properties can be used within this context? Context can be an data receipt function, data creation function, API, etc.

47.1 Simple Example

48 Default Values

requirements:

- R-31-DEFAULT-VALUES-OF-RDF-OBJECTS
- R-38-DEFAULT-VALUES-OF-RDF-LITERALS

48.1 Description

- default values for RDF literals
- default values for RDF objects
- default values are inferred automatically
- It should be possible to declare the default value for a given property, e.g. so that input forms can be pre-populated and to insert a required property that is missing in a web service call.

48.2 Simple Example

Jedis have only 1 blue laser sword per default. Siths, in contrast, normally have 2 red laser swords.

```
# rule (SPIN)
# -----------
owl:Thing
    spin:rule [
        a sp:Construct ;
            sp:text """
                CONSTRUCT {
                    ?this :laserSwordColor "blue"^^xsd:string ;
                    ?this :numberLaserSwords "1"^^xsd:nonNegativeInteger .
                }
                WHERE {
                    ?this a :Jedi .
                } """ ; ] .
owl:Thing
    spin:rule [
        a sp:Construct ;
            sp:text """
                CONSTRUCT {
                    ?this :laserSwordColor "red"^^xsd:string ;
```

```
                    ?this :numberLaserSwords "2"^^xsd:nonNegativeInteger .
                }
                WHERE {
                    ?this a :Sith .
                } """ ; ] .
```

```
# data
# ----
:Joda a :Jedi .
:DarthSidious a :Sith .
```

```
# inferred triples
# ----------------
:Joda
    :laserSwordColor "blue"^^xsd:string ;
    :numberLaserSwords "1"^^xsd:nonNegativeInteger .
:DarthSidious
    :laserSwordColor "red"^^xsd:string ;
    :numberLaserSwords "2"^^xsd:nonNegativeInteger .
```

49 Mathematical Operations

requirements:

- R-42-MATHEMATICAL-OPERATIONS
- R-41-STATISTICAL-COMPUTATIONS

49.1 Description

examples:

- adding 2 dates
- add number of days to start date
- area = width * height
- Statistical Computations: average, mean, sum

49.2 Simple Example

50 Language Tag Matching

requirement:

- R-47-LANGUAGE-TAG-MATCHING

51 Cardinality on Literal Language Tags

requirements:

- R-49-RDF-LITERALS-HAVING-AT-MOST-ONE-LANGUAGE-TAG
- R-48-MISSING-LANGUAGE-TAGS

51.1 Description

51.2 Example

check that no language is used more than once per property

```
# DQTP
# ----
SELECT DISTINCT ?s WHERE { ?s %%P1%% ?c
    BIND ( lang(?c) AS ?l )
    FILTER (isLiteral (?c) && lang(?c) = %%V1%%)}
GROUP BY ?s HAVING COUNT (?l) > 1
```

- ONELANGPattern [**?**]
- A literal value should contain at most 1 literal for a language.
- P1 is the property containing the literal and V1 is the language we want to check.

```
# test binding (DQTP)
---------------------
P1 => foaf:name
V1 => en
```

- a single English ("en") foaf:name

```
# valid data
# ----------
:LeiaSkywalker
    foaf:name 'Leia Skywalker'@en .
```

```
# invalid data
# ------------
:LeiaSkywalker
    foaf:name 'Leia Skywalker'@en ;
    foaf:name 'Leia'@en .
```

52 Whitespace Handling

requirement:

- R-50-WHITESPACE-HANDLING-OF-RDF-LITERALS

52.1 Description

- avoid whitespaces in literals neither leading nor trailing white spaces

52.2 Simple Example

53 HTML Handling

requirement:

- R-51-HTML-HANDLING-OF-RDF-LITERALS

  description:

- check if there are no html tags included in literals (of specific data properties within the context of specific classes)

53.1 Simple Example

54 Required Properties

requirements:

- R-68-REQUIRED-PROPERTIES

54.1 Description

54.2 Simple Example

55 Optional Properties

requirements:

- R-69-OPTIONAL-PROPERTIES

55.1 Description

55.2 Simple Example

56 Repeatable Properties

requirements:

- R-70-REPEATABLE-PROPERTIES

56.1 Description

56.2 Simple Example

57 Conditional Properties

requirements:

- R-71-CONDITIONAL-PROPERTIES

57.1 Description

- various conditions possible
- condition: universal quantification on object and data properties
- condition: existential quantification on object and data properties
- condition: if specific properties are present, then specific other properties also have to be present

57.2 Simple Example

58 Recommended Properties

requirements:

- R-72-RECOMMENDED-PROPERTIES

58.1 Description

58.2 Simple Example

59 Negative Property Constraints

requirements:

- R-52-NEGATIVE-OBJECT-PROPERTY-CONSTRAINTS
- R-53-NEGATIVE-DATA-PROPERTY-CONSTRAINTS

59.1 Description

- instances of a specific class must not have some object property
- OWL 2 DL: ObjectComplementOf (ObjectSomeValuesFrom (ObjectPropertyExpression owl:Thing))

59.2 Example

```
# ShEx
# ----
<FeelingForce> {
    :feelingForce (true) ,
    :attitute xsd:string }
<JediMentor> {
    :feelingForce (true) ,
    :attitute ('good') ,
    :laserSwordColor xsd:string ,
    :numberLaserSwords xsd:nonNegativeInteger ,
    :mentorOf @<JediStudent> ,
   !:studentOf @<JediMentor> }
<JediStudent> {
    :feelingForce (true) ,
    :attitute ('good') ,
    :laserSwordColor xsd:string ,
    :numberLaserSwords xsd:nonNegativeInteger ,
   !:mentorOf @<JediStudent> ,
    :studentOf @<JediMentor> }
```

- A matching triple has any predicate except those excluded by the '!' operator.
- http://w3.org/brief/Mzk0

```
# individuals matching 'FeelingForce' and 'JediMentor' data shapes
# ----------------------------------------------------------------
:Obi-Wan
    :feelingForce true ;
    :attitute 'good' ;
    :laserSwordColor 'blue' ;
    :numberLaserSwords '1'^^xsd:nonNegativeInteger ;
    :mentorOf :Anakin .
```

```
# individuals matching 'FeelingForce' and 'JediStudent' data shapes
# ----------------------------------------------------------------
:Anakin
    :feelingForce true ;
    :attitute 'good' ;
    :laserSwordColor 'blue' ;
    :numberLaserSwords '1'^^xsd:nonNegativeInteger ;
    :studentOf :Obi-Wan .
```

60 Handle RDF Collections

requirement:

- R-120-HANDLE-RDF-COLLECTIONS

60.1 Description

examples:

- size of collection
- first / last element of list must be a specific literal
- compare elements of collection
- are collections identical?
- actions on RDF lists: http://www.snee.com/bobdc.blog/2014/04/rdf-lists-and-sparql.html
- 2. list element equals ”
- Does the list have more than 10 elements?

60.2 Example

- get 2. list element

```
# SPIN
# ----
:getListItem
    a spin:Function ; rdfs:subClassOf spin:Functions ;
    spin:constraint [
        rdf:type spl:Argument ;
        spl:predicate sp:arg1 ;
        spl:valueType rdf:List ;
        rdfs:comment "list" ; ] ;
    spin:constraint [
        rdf:type spl:Argument ;
        spl:predicate sp:arg2 ;
```

```
        spl:valueType xsd:nonNegativeInteger ;
        rdfs:comment "item position (starting with 0)" ; ] ;
    spin:body [
        a sp:SELECT ;
        sp:text """
            SELECT ?item
            WHERE {
                ?arg1 :contents/rdf:rest{?arg2}/rdf:first ?item } """ ; ] ;
    spin:returnType rdfs:Resource .
```

```
# data
# ----
:Jinn :students
     ( :Xanatos :Kenobi ) .
```

```
# SPIN
# ----
BIND ( :getListItem( ?list, "1"xsd:nonNegativeInteger ) AS ?listItem ) .
```

- SPIN function call
- retrieves the 2. item from the list (2. student of Jedi mentor Jinn)

```
# result
# ------
:Kenobi
```

61 Recursive Queries

requirements:

- R-222-RECURSIVE-QUERIES

61.1 Description

- If we want to define Resource Shapes, remember that it is a recursive language (the valueShape of a Resource Shape is in turn another Resource Shape). There is no way to express that in SPARQL without hand-waving "and then you call the function again here" or "and then you embed this operation here" text. The embedding trick doesn't work in the general case because SPARQL can't express recursive queries, e.g. "test that this Issue is valid and all of the Issues that references, recursively".
- Most SPARQL engines already have functions that go beyond the official SPARQL 1.1 spec. The cost of that sounds manageable to me.

61.2 Simple Example

```
# ShEx
# ----
<IssueShape> {
    :state (:unassigned :assigned),
    :reportedBy @<UserShape>,
    :related @<IssueShape>*
}
```

62 Value is Valid for Datatype

requirements:

–

62.1 Description

– make sure that a value is valid for its datatype

62.2 Simple Example

– a date is really a date
– SPARQL regex can be used

62.3 Simple Example

```
# SPIN
# ----
FILTER ( datatype( ?shoeSize ) = xsd:nonNegativeInteger )
isNumeric ( ?shoeSize )
```

– datatype of ?showSize id xsd:nonNegativeInteger
– datatype is really numeric

63 Individual Equality

63.1 Description

– states that two different names are known to refer to the same individual [**?**]

63.2 Simple Example

DL:
julia = john

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	{julia}	-	-	{john}	=	
class	{john}	-	-	{julia}	=	

64 Individual Inequality

64.1 Description

64.2 Simple Example

DL:
julia $\neq$ john

– asserts that Julia and John are actually different individuals

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	{julia}	-	-	{john}	$\neq$	
class	{john}	-	-	{julia}	$\neq$	

65 Equivalent Properties

65.1 Description

– An equivalent object properties axiom EquivalentObjectProperties(OPE1 ... OPEn) states that all of the object property expressions OPEi, 1 $\leqslant$ i $\leqslant$ n, are semantically equivalent to each other. This axiom allows one to use each OPEi as a synonym for each OPEj — that is, in any expression in the ontology containing such an axiom, OPEi can be replaced with OPEj without affecting the meaning of the ontology. The axiom EquivalentObjectProperties(OPE1 OPE2) is equivalent to the following two axioms SubObjectPropertyOf(OPE1 OPE2) and SubObjectPropertyOf(OPE2 OPE1).
– An equivalent data properties axiom EquivalentDataProperties(DPE1 ... DPEn) states that all the data property expressions DPEi, 1 $\leqslant$ i $\leqslant$ n, are semantically equivalent to each other. This axiom allows one to use each DPEi as a synonym for each DPEj — that is, in any expression in the ontology containing such an axiom, DPEi can be replaced with DPEj without affecting the meaning of the ontology. The axiom EquivalentDataProperties(DPE1 DPE2) can be seen as a syntactic shortcut for the following axiom SubDataPropertyOf(DPE1 DPE2) and SubDataPropertyOf(DPE2 DPE1).

65.2 Simple Example

```
1 # OWL 2
2 # -----
3 :hasBrother owl:equivalentProperty :hasMaleSibling .
4 :Chris :hasBrother :Stewie .
5 :Stewie :hasMaleSibling :Chris .
```

entailments:

```
:Chris :hasMaleSibling a:Stewie .
:Stewie :hasBrother a:Chris .
```

DL:
:hasBrother ⊑ :hasMaleSibling ⊓ :hasMaleSibling ⊑ :hasBrother

66 Property Assertions

66.1 Description

- positive property assertions
- negative property assertions
- A positive object property assertion ObjectPropertyAssertion(OPE a1 a2) states that the individual a1 is connected by the object property expression OPE to the individual a2.
- A negative object property assertion NegativeObjectPropertyAssertion(OPE a1 a2) states that the individual a1 is not connected by the object property expression OPE to the individual a2.
- A positive data property assertion DataPropertyAssertion(DPE a lt) states that the individual a is connected by the data property expression DPE to the literal lt.
- A negative data property assertion NegativeDataPropertyAssertion(DPE a lt) states that the individual a is not connected by the data property expression DPE to the literal lt.

66.2 Simple Example

```
# OWL 2
# -----
NegativeObjectPropertyAssertion( :hasSon :Peter :Meg )
```

- Meg is not a son of Peter.

DL:

Thomas: ToDo: DL

66.3 Simple Example

```
# OWL 2
# -----
DataPropertyAssertion( :hasAge :Meg "17"^^xsd:integer )
```

- Meg is seventeen years old.

DL:
:hasAge (:Meg , "17"8sd:integer)

67 Functional Properties

67.1 Description

– An object property functionality axiom FunctionalObjectProperty(OPE) states that the object property expression OPE is functional — that is, for each individual x, there can be at most one distinct individual y such that x is connected by OPE to y. Each such axiom can be seen as a syntactic shortcut for the following axiom: SubClassOf(owl:Thing ObjectMaxCardinality(1 OPE)).

67.2 Simple Example

```
# OWL 2
# -----
:hasFather rdf:type owl:FunctionalProperty .
:Stewie :hasFather :Peter .
:Stewie :hasFather :Peter_Griffin .
```

– Each object can have at most one father.

entailment:

```
:Peter_Griffin owl:sameAs :Peter .
```

DL:
owl:Thing $\sqsubseteq \leqslant 1$:hasFather

68 Inverse-Functional Properties

68.1 Description

– An object property inverse functionality axiom InverseFunctionalObject-Property(OPE) states that the object property expression OPE is inverse-functional — that is, for each individual x, there can be at most one individual y such that y is connected by OPE with x. Each such axiom can be seen as a syntactic shortcut for the following axiom: SubClassOf(owl:Thing ObjectMaxCardinality(1 ObjectInverseOf(OPE))).

68.2 Simple Example

```
# OWL 2
# -----
:fatherOf rdf:type owl:InverseFunctionalProperty .
:Peter :fatherOf :Stewie .
:Peter_Griffin :fatherOf :Stewie .
```

– Each object can have at most one father.

entailment:

```
:Peter owl:sameAs :Peter_Griffin .
```

DL:

Thomas: ToDo: DL

69 Value Restrictions

69.1 Description

- Individual Value Restrictions: A has-value class expression ObjectHasValue(OPE a) consists of an object property expression OPE and an individual a, and it contains all those individuals that are connected by OPE to a. Each such class expression can be seen as a syntactic shortcut for the class expression ObjectSomeValuesFrom(OPE ObjectOneOf(a)).
- Literal Value Restrictions: A has-value class expression DataHasValue(DPE lt) consists of a data property expression DPE and a literal lt, and it contains all those individuals that are connected by DPE to lt. Each such class expression can be seen as a syntactic shortcut for the class expression DataSomeValuesFrom(DPE DataOneOf(lt)).

69.2 Simple Example

```
:Peter :fatherOf :Stewie .
ObjectHasValue( :fatherOf :Stewie )
```

- The has-value class expression contains those individuals that are connected through the :fatherOf property with the individual :Stewie
- :Peter is classified as its instance

DL:

Thomas: ToDo: DL

70 Self Restrictions

70.1 Description

- A self-restriction ObjectHasSelf(OPE) consists of an object property expression OPE, and it contains all those individuals that are connected by OPE to themselves.

70.2 Simple Example

```
:Peter :likes :Peter .
ObjectHasSelf( :likes )
```

- The self-restriction contains those individuals that like themselves
- :Peter is classified as its instance

71 Data Property Facets

requirement:

- R-46-CONSTRAINING-FACETS

71.1 Description

- For datatype properties it should be possible to declare frequently needed "facets" to drive user interfaces and validate input against simple conditions, including min/max value, regular expressions, string length etc. similar to XSD datatypes.
- constraining facets to restrict datatypes of RDF literals
- constraining facets: xsd:length, xsd:minLength, xsd:maxLength, xsd:pattern, xsd:enumeration, xsd:whiteSpace, xsd:maxInclusive, xsd:maxExclusive, xsd:minExclusive, xsd:minInclusive, xsd:totalDigits, xsd:fractionDigits

71.2 Simple Example

string matches regular expression

```
# OWL 2 QL (functional-style syntax)
# ----------------------------------
Declaration( Datatype( :SSN ) )
DatatypeDefinition(
    :SSN
    DatatypeRestriction( xsd:string xsd:pattern "[0-9]{3}-[0-9]{2}-[0-9]{4}" ) )
DataPropertyRange( :hasSSN :SSN )
```

```
# OWL 2 QL (turtle syntax)
# ------------------------
:SSN
    a rdfs:Datatype ;
    owl:equivalentClass [
        a rdfs:Datatype ;
        owl:onDatatype xsd:string ;
        owl:withRestrictions (
            [ xsd:pattern "[0-9]{3}-[0-9]{2}-[0-9]{4}" ] ) ] .
:hasSSN rdfs:range :SSN .
```

- A social security number is a string that matches the given regular expression. The second axiom defines :SSN as an abbreviation for a datatype restriction on xsd:string. The first axiom explicitly declares :SSN to be a datatype. The datatype :SSN can be used just like any other datatype; for example, it is used in the third axiom to define the range of the :hasSSN property.

```
# valid data
# ----------
:TimBernersLee
    :hasSSN "123-45-6789"^^:SSN .
```

```
# invalid data
# ------------
:TimBernersLee
    :hasSSN "123456789"^^:SSN .
```

c. type	context concept	left p. list	right p. list	concepts	c. element	c. value
class	SSN	-	-	xsd:string	xsd:pattern	'[0-9]3-[0-9]2-[0-9]4'

72 Primary Key Properties

requirement:

–

72.1 Description

– It is often useful to declare a given (datatype) property as the "primary key" of a class, so that the system can enforce uniqueness and also automatically build URIs from user input and data imported from relational databases or spreadsheets.

72.2 Simple Example

– Each object is uniquely identified by its social security number.

```
# DSP
# ---
-
```

```
# OWL2 DL
# -------
owl:Thing owl:hasKey ( :hasSSN ) .
```

```
# ReSh
# ----
-
```

```
# ShEx
# ----
-
```

– The first axiom makes :hasSSN the key for instances of the owl:Thing class; thus, only one individual can have a particular value for :hasSSN.

```
# data
# ----
:Peter :hasSSN "123-45-6789" .
:PeterGriffin :hasSSN "123-45-6789" .
```

– Since the values of :hasSSN are the same for the individuals a:Peter and a:PeterGriffin, these two individuals are equal - that is, this ontology entails the following assertion :Peter owl:sameAs :PeterGriffin .

DL:

Thomas: ToDo: DL

73 Constraint Language Mapping into DL

The following table shows the mapping between constraint language and DL:

Table 1. Constraint Language Mapping into DL

Constraint Type	**Constraint Language Expression**	**DL Expression**	*DL-Lite*$_{\mathcal{A}}$

property groups	`<Human> {`
	`  (`
	`  foaf:name xsd:string |`
	`  foaf:givenName xsd:string`
	`  ),`
	`  foaf:mbox IRI }`
		$Human \sqsubseteq \exists$foaf:name $\sqcup \exists$foaf:givenName
		$Human \sqsubseteq \exists foaf{:}mbox$
		$\exists$foaf:name $\sqsubseteq \neg\exists$foaf:givenName
		ρ(foaf:name) $\sqsubseteq$ `xsd:string`
		ρ(foaf:givenName) $\sqsubseteq$ `xsd:string`
		$\exists foaf{:}name^{-} \sqsubseteq IRI$
			false

enumeration / value set			
R30 - Subject	...	$\exists\, dcterms \sqsubseteq ComputerScience \sqcup SocialScience \sqcup Librarianship$	false
R30 - Object	...	$\exists\, dcterms^{-} \sqsubseteq ComputerScience \sqcup SocialScience \sqcup Librarianship$	false
R37	...	???	
200	...	ρ(foaf:givenName) $\sqsubseteq \neg$`xsd:integer`	true
R33	...	$\exists\, attitude^{-} \sqsubseteq \neg Color$	true
exclusive OR	?	$Human \sqsubseteq Man \sqcup Woman$ $Man \sqsubseteq \neg Woman$	false
exclusive OR (variant)	?	$Man \sqcup Woman \sqsubseteq Human$ $Man \sqsubseteq \neg Woman$	true