Expressivity and Effects on Complexity of RDF Constraint Languages

XXXXX[1] and XXXXX[2]

[1] XXXXX
XXXXX,
[2] XXXXX
XXXXX

Abstract. **Keywords:** ..

1 Motivation

For many RDF applications, the formulation of constraints and the automatic validation of data according to these constraints is a much sought-after feature. In 2013, the W3C invited experts from industry, government, and academia to the RDF Validation Workshop[3], where first use cases have been presented and discussed. Two WGs, that follow up on this workshop and address RDF constraint formulation and validation, are established in 2014: the W3C RDF Data Shapes WG[4] and the DCMI RDF Application Profiles WG[5].

Thomas: ToDo for Thomas: RDF validation requirements database

There are long and controversial discussions in these WGs if or if not OWL should be used RDF validation when assuming closed world semantics. OWL is an instantiation of a DSCL which is high-level, human-friendly (human-readable and human-understandable), fairly concise, and very expressive. There are lots of benefits but also a huge amount of drawbacks when using OWL for the purpose to formulate and to validate RDF constraints.

– explain why specific OWL 2 constructs could be used for RDF validation
– explain why specific OWL 2 constructs should not be used for RDF validation

For RDF, SPARQL is generally seen as the method of choice to validate data according to certain constraints, although it is not ideal for their formulation. In contrast, OWL 2 DL constraints are comparatively easy to understand, but lack an implementation to validate RDF data. Within our developed SPIN[6] validation environment, we fully implemented an automatic validation of all OWL 2 DL constructs. The implementation can be tested at http://purl.org/net/rdfval-demo and the OWL 2 SPIN mapping is maintained at https://github.com/boschthomas/OWL2-SPIN-Mapping.

Andy: IMHO this should be decribed more precisely because validation in terms of inconsistency detection is still available, not sure about the completeness for OWL 2 DL

Andy: you should mention at this point that this validation is in terms of RDF validation, because inconsistency can't be done completely

Thomas: ToDo Thomas: constraint languages

[3] http://www.w3.org/2012/12/rdf-val/
[4] http://www.w3.org/2014/rds/charter
[5] http://wiki.dublincore.org/index.php/RDF-Application-Profiles
[6] http://spinrdf.org/

Constraint Validation with Reasoning

Constraint validation does not appear to be part of the services provided by OWL. This has lead to claims that OWL cannot be used for constraint validation. However reasoning, which is the core service provided by OWL, and constraint validation are indeed very closely related.

Andy: a more formal example would be nice! ;)

Reasoning is the process of determining what follows from what has been stated. Reasoning ranges from simple (students are people, John is a student, therefore John is a person) to the very complex. Reasoning can also recognize impossibilities (students are people, John is a student, John is not a person, therefore there is a contradiction).

– inferencing as a pre validation step
– both should be possible: (1) constraint validation with reasoning and (2) constraint validation without reasoning

Andy: we should mention a reason why we need this...

With regard to typing, W3C is assuming (AFAIK) that all types will be explicit in the instance data - for the purposes of validation. There is nothing, however, to prevent an application from running reasoning prior to validation to add inferred triples to the data being validated. But the general feeling right now is that validation acts on instance data without requiring that the validator apply reasoning in order to do so.

We answer the following **research questions**:

1. which constraints are expressible by DL?
2. which constraints are not expressible by DL but by a query language such as Datalog or SPARQL?
3. for which requirements formulating RDF constraints the expressivity of DL-Lite$_A$ respectively OWL 2 QL is sufficient?
4. for which requirements additional constraint languages are needed to express related constraints?
5. which constraint languages are suitable to express these constraints?
6. what are the effects on complexity to express these constraints with and without OWL 2 QL and OWL 2 DL inferencing?
7. what are the effects on complexity of constraints expressible by DL?
8. what are the effects on complexity of constraints not expressible by DL but by a query language?

Contributions of this paper are:

1.
2.
3. We evaluated to which extend the five possible standard constraint languages fulfill each requirement.

Ideas

– RDF validation using OWL 2 QL reasoning by SPARQL query expansion

- if using OWL 2 DL as constraint language or using constraints equivalent to OWL 2 QL you can use reasoning
- reasoning can also be executed using SPARQL query expansion.
- reasoning not executing using reasoner

- complete with reasoning — OW
- complete without reasoning — CW
- there is no query rewriting mechanism for OWL 2, just for OWL 2 QL
- show that OWL 2 QL and further constraint languages together are complete

2 RDF Validation Requirements and Reasoning

I would explain the addressed problem by an example, like the following: consider the KB $\mathcal{K} = \{Person \sqsubseteq \exists hasAncestor.Person, Person(\mathbf{Thomas})\}$, using the OWA we know that **Thomas** has some Ancestor, but we do not know them, so $\mathcal{K}$ is consistent. In CWA the constraint needs to be satisfied only by named individuals, so there are some validation errors.
A more complex example could be:
Assume $\mathcal{K} = \{Person \sqsubseteq \exists hasAncestor.Person, (\mathsf{funct}\ hasAncestor), Person(\mathbf{Thomas}), hasAncestor(\mathbf{Thoma}$
in CWA there is a clash because of the functional role *hasAncestor* and the UNA, but in OWA there is no UNA by default so the reasoner will produce that $\mathbf{Erman} \equiv \mathbf{Andy}$. But this is not a problem if we add UNA to the OWA, like in the *DL-Lite* family. Another point would be that if we perform RDF validation with no reasoning, we will get a further clash **Erman** is not defined to be a *Person*, but if we do the validation after some reasoning, there will be no clash, because by definition the following definition will be inferred: $Person(\mathbf{Erman})$. So in my understanding it would be more useful/reasonable to perform the validation task after reasoning.

Thomas: ToDO Thomas: RDF Validation Requirements and Inferencing

2.1 OWL 2 QL Reasoning

Thomas: ToDo for Andy and Erman: may you shorten the next 3 paragraphs? What is needed for the paper?

OWL 2 profiles (or fragments in logic literature) are restricted (sublanguage) versions of OWL 2 that offers different trade-offs regarding expressivity vs. efficiency in reasoning. There are three main profiles of OWL 2, which are OWL 2 QL, OWL 2 RL and OWL 2 EL namely, each designed to be useful for different purposes and application scenarios. The choice of which profile to use is purpose-specific on what to express and to perform the reasoning about. We refer reader to [4] for a comprehensive treatment of those profiles.

OWL 2 QL is an OWL 2 profile which focuses on reasoning on query answering with very large size of instance data. The acronym QL stands for query language as query answering in this profile can be done by rewriting queries into a standard relational query language (also known as First-Order Rewritability). Also, conjunctive query answering can be done by using conventional relational database systems. Aiming at efficiency on working with large size data, the expressive power of the profile is quite limited as expected. As a result of this, reasoning in OWL 2 QL is highly efficient so that sound and complete conjunctive query answering can be performed in LogSpace in the size of the data assertions as well as the ontology consistency and class expression subsumption can be performed in polynomial time.

OWL 2 QL is based on the DL-Lite family of description logics, a tractable family of fragments of first-order logic [1, 3]. However, an important difference between the DL-Lite family and OWL is the unique name assumption (UNA). UNA is common in data management, therefore it is adopted in the DL-Lite family whereas not adopted in OWL. In OWL, one uses constructs **sameAs** or **differentFrom** explicitly to state that two individuals, say a and b, are the *same* or different respectively. For that reason in OWL 2 QL, any construct e.g., number restrictions or functionality constraints which can interfere with UNA and also which can cause higher complexity without the UNA, has been avoided.

Among the members of DL-Lite family, DL-Lite$_R$ is the one that OWL 2 QL is based on. The reason is to avoid any problematic issue which might appear in the explicit axiomatization of UNA, since DL-Lite$_\mathcal{R}$ in general does not require the UNA, because making this assumption has no semantic affect on a DL-Lite$_R$ ontology.

Andy: but for

Subsumption. Subsumption relationships of both classes and properties are part of basic reasoning.

:Jedi $\sqsubseteq$:FeelingForce
:JediMaster $\sqsubseteq$:Jedi

All Jedi masters are Jedis feeling the force. These sub-class relationships can also be expressed by OWL 2 QL:

```
:Jedi rdfs:subClassOf :FeelingForce .
:JediMaster rdfs:subClassOf :Jedi .
```

Valid data must contain the three class assignments:

```
:Yoda a :JediMaster .
:Yoda a :Jedi .
:Yoda a :FeelingForce .
```

These class assignments can be either explicitly stated or implicitly inferred when reasoning is performed before actually validating the data.

When only the triple `:Yoda a :JediMaster` is given and reasoning is not executed, then a constraint violation is raised. After reasoning, however, the triples `:Yoda a :Jedi , :FeelingForce` are inferred resulting in valid data.

Property Domain. The property domain constraint

$\exists$:studentOf . $\top$ $\sqsubseteq$:JediStudent

restricts that individuals having :studentOf relationships must be Jedi students. This property domain constraint can also be expressed by OWL 2 QL:

```
:studentOf rdfs:domain :JediStudent .
```

Without reasoning, the data `:Anakin :studentOf :Obi-Wan` is invalid and causes a constraint violation, as it is not explicitly stated that `:Anakin` is assigned to the class `:Jedi`. When inferencing is performed before validating, the class assignment `:Anakin rdf:type :Jedi` is inferred which prevents the constraint violation to be raised.

2.2 OWL 2 DL Reasoning

The Semantic Web ontology language OWL 2 DL [7] was standardized by the World Wide Web Consortium (W3C) in 2009 (and updated in 2012) as a description logic-like formalism. OWL 2 DL has high expressivity, yet maintains decidability for main reasoning tasks e.g., ontology satisability, entailment checking. The drawback of its expressive power results as a lack of computational efficiency in performance. In general, reasoning in OWL 2 DL is in N2EXPTIME [4].

As a result of its expressive power, OWL 2 DL allows a large variety of sophisticated modeling capabilities for many application domains. On the other hand, to maintain the decidability, OWL 2 DL has a numerous ways of syntactic restrictions. One example is that OWL 2 DL allows expressing transitive properties as well as asymmetric properties, yet a property to be both transitive and asymmetric (just as ancestor relation) is not allowed. In the sequel, from RDF validation perspective, we will give some examples of constraints that remains at the outside of its scope.

Existential Quantification.

∃ :hasLaserSword . :LaserSword

This existential quantification contains all those individuals that are connected by :hasLaserSword to an individual that is an instance of the class :LaserSword. In OWL 2 DL, this existential quantification can also be expressed:

```
[ a owl:Restriction ;
  owl:onProperty :hasLaserSword ;
  owl:someValuesFrom :LaserSword ;
  rdfs:subClassOf :Jedi ] .
```

When our data contains the triple `:Luke :hasLaserSword :BlueLaserSword` and we perform reasoning, we can infer that `:Luke` is a :Jedi, as he has a laser sword. As the individual `:Luke` is assigned to the class :Jedi, all constraint associated with this class are also validated, for example that Jedis must have blue laser swords. Without reasoning, these constraints won't be validated as `:Luke` is not within the class extension of `:Jedi`.

[7] See http://www.w3.org/TR/owl2-direct-semantics/.

3 RDF Validation Requirements

For the majority of the requirements to formulate RDF constraints reasoning is not related to the constraints and is therefore not performed prior to the validation.

Andy: If we do reasoning by query rewriting in terms of backward chaining we don't need to distinguish between them, because if you run a constraint validation query through an engine like –ontop– (OBDA tool) or ELITE (federated query engine), the engine will only do some rewritings if it is possible, and if you just check e.g. pattern matching there will be no rewriting...

Andy: IMHO literal pattern matching is not expressible in DL!

3.1 Expressible by OWL 2 QL

Literal Pattern Matching. There are multiple use cases associated with the requirement to match literals according to given patterns[8]. A social security number, for example, is a string that matches a given regular expression - which can be expressed by OWL 2 QL:

```
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 .
```

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 like in the third axiom to define the range of the `hasSSN` property. The literal pattern matching constraint above validates `hasSSN` literals according to the stated regular expression causing a constraint violation for the triple `TimBernersLee hasSSN "123456789"^^SSN`, but not for the triple `TimBernersLee hasSSN "123-45-6789"^^SSN`.

3.2 Not Expressible by OWL 2 QL but by OWL 2 DL

Allowed Values[9]. 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 resources). This is often rendered in drop down boxes or radio buttons in user interfaces. The constraint 'Jedis can only have blue, green, or white laser swords' can be expressed by OWL 2 DL, DSP, ReSh, ShEx, SPIN, and SPARQL.

$$\text{Jedi} \equiv \exists\ \text{laserSwordColor}\ .\ \{\text{blue, green, white}\}$$

In DSP, the constraint does not look that concise:

```
personDescriptionTemplate
    a dsp:DescriptionTemplate ;
    dsp:resourceClass Jedi ;
    dsp:statementTemplate [
```

[8] corresponds to R-44-PATTERN-MATCHING-ON-RDF-LITERALS

[9] corresponds to R-30-ALLOWED-VALUES-FOR-RDF-OBJECTS and R-37-ALLOWED-VALUES-FOR-RDF-LITERALS

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

When representing the constraint by OWL2 DL a new datatype is defined by simply enumerating its literals:

Andy: One constraint language is IMHO enought... Maybe not!?

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

ReSh can also be used for this purpose:

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

ShEx is the most intuitive and concise means representing the allowed values constraint:

```
Jedi {
    laserSwordColor ('blue' 'green' 'white') }
```

Data containing the triples `Yoda a Jedi ; laserSwordColor 'blue'` is valid, whereas data including the triples `DarthMaul a Jedi ; laserSwordColor 'red'` is invalid.

Context-Specific Exclusive OR of Properties[10]. An individual can either have a relationship via property A or via property B, but not both. To take an example, a Jedi is either attacking by sword or by force but not both (such a fight would not be fair):

$$\text{:Jedi} \sqsubseteq (\neg A \sqcap B) \sqcup (A \sqcap \neg B)$$
$$A \equiv \forall \text{ :attackingBySword . xsd:boolean}$$
$$B \equiv \forall \text{ :attackingByForce . xsd:boolean}$$

Andy: IMHO: If you use the universal quantifier it is also valid that a Jedi neither have attackingBySword nor attackingByForce, so it should be an existential quantifier...

Context-specific exclusive OR of properties can be expressed by ShEx. In this case, the constraint context is the class `:Jedi`.

```
:Jedi { (
    :attackingBySword xsd:boolean |
    :attackingByForce xsd:boolean ) }
```

Andy: Do you mean DL or QL at this point?

The ~~same constraint may be~~ expressed by OWL 2 DL , but not very intuitively and concisely. This can rather be seen as a workaround, as anonymous classes are built for each of the exclusive properties.

```
:Jedi owl:disjointUnionOf ( :CC1 :CC2 ) .
:CC1 rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :attackingBySword ;
    owl:allValuesFrom xsd:boolean ] .
:CC2 rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :attackingByForce ;
    owl:allValuesFrom xsd:boolean ] .
```

Andy: Formulars in DL would be nice... ;)

Considering the following data, Luke as well as Darth Sidious are associated with the class `:Jedi`. As Darth Sidious has both relationships, a constraint Violation is raised.

```
:Luke
    a :Jedi ;
    :attackingBySword true .
:DarthSidious
    a :Jedi ;
    :attackingByForce true ;
    :attackingBySword true ;
```

3.3 Not Expressible by OWL 2 DL But by Other Constraint Languages

The majority of constraints can neither be expressed by OWL 2 QL nor by OWL 2 DL. In such cases, these constraints are represented by other constraint languages like DSP, ShEx, ReSh, or SPIN.

Default Values. It should be possible to declare the default value for a given object and data 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. Siths have per default two red laser swords. If there is only stated that Darth Maul is a Sith (`:DarthMaul a :Sith .`), then additional default triples should be inferred automatically:

```
:DarthMaul
    :laserSwordColor "red"^^xsd:string ;
    :numberLaserSwords "2"^^xsd:nonNegativeInteger .
```

The default values constraint can only be expressed by ReSh, SPIN, and SPARQL. In SPIN, we can define a rule associated with the class `owl:Thing`. This rule is applicable for each resource, as each resource is implicitly of the type `owl:Thing`.

[10] corresponds to R-11-CONTEXT-SPECIFIC-EXCLUSIVE-OR-OF-PROPERTIES

```
owl:Thing spin:rule [ a sp:Construct ; sp:text """
    CONSTRUCT {
        ?this :laserSwordColor "red"^^xsd:string ;
              :numberLaserSwords "2"^^xsd:nonNegativeInteger . }
    WHERE {
        ?this a :Sith . } """ ; ] .
```

For each resource, the SPARQL CONSTRUCT query within the rule is executed creating the default triples. ReSh can also be used to express default values:

```
:Sith a rs:ResourceShape ;
    rs:property [
        rs:name "laserSwordColor" ;
        rs:propertyDefinition :laserSwordColor ;
        rs:valueType xsd:string ;
        rs:defaultValue "red"^^xsd:string
        rs:occurs rs:Exactly-one ; ] .
```

4 Evaluation

We evaluated to which extend the five possible standard constraint languages fulfill each requirement to formulate RDF constraints. Tilde means that this constraint may be fulfilled by that particular constraint language - either by limitations, workarounds, or extensions. We also evaluated if a specific constraint is fulfilled by OWL 2 QL or if the more expressive OWL 2 DL is needed. Inferencing may be performed prior to validating constraints. This is marked with an asterisk.

We divide these constraints into three classes:

– constraints expressible by OWL 2 QL
– constraints not expressible by OWL 2 QL but by OWL 2 DL
– constraints only expressible by other DSCLs

Bosch et al. explained each requirement and associated constraint in detail and give at least one example for each of them[2]. In order to logically underpin each requirement, we added mappings to DL.

Constraints Expressible by OWL 2 QL.

Thomas: How should we publish this? ordering of author not determined so far

constraint	**DSP**	**OWL2-DL**	**OWL2-QL**	**ReSh**	**ShEx**	**SPIN**
*Subsumption	✗	✓	✓	∼	✓	✓
*Class Equivalence	✗	✓	✓	✗	✗	✓
*Sub Properties	✗	✓	✓	✗	✗	✓
*Property Domain	✗	✓	✓	✗	✗	✓
*Property Range	✗	✓	✓	✗	✗	✓
*Inverse Object Properties	✗	✓	✓	∼	✗	✓
*Symmetric Object Properties	✗	✓	✓	✗	✗	✓
*Asymmetric Object Properties	✗	✓	✓	✗	✗	✓
*Reflexive Object Properties	✗	✓	✓	✗	✗	✓
*Irreflexive Object Properties	✗	✓	✓	✗	✗	✓
Disjoint Properties	✗	✓	✓	✗	✗	✓
Disjoint Classes	✗	✓	✓	✗	✗	✓
Context-Sp. Property Groups	✗	∼	∼	✓	✓	✓
Context-Sp. Inclusive OR of P.	✗	∼	∼		✗	✓
Context-Sp. Inclusive OR of P. Groups	✗	∼	∼		✗	✓
Recursive Queries	✓	✓	✓	✓	✓	∼
Individual Inequality	✗	✓	✓	✗	✗	✓
*Equivalent Properties	✗	✓	✓	✗	✗	✓
Property Assertions	✗	✓	∼	✗	✗	✓
Data Property Facets	✗	✓	✓	✗	✗	✓
Literal Pattern Matching	✗	✓	✓	✓	✓	✓
Negative Literal Pattern Matching	✗	✓	✓			✓

Constraints not Expressible by OWL 2 QL but by OWL 2 DL.

constraint	**DSP**	**OWL2-DL**	**OWL2-QL**	**ReSh**	**ShEx**	**SPIN**
*Object Property Paths	✗	✓	✗	✗	✗	✓
*Intersection	✗	✓	✗			✓
*Disjunction	✗	✓	✗			✓
*Negation	✗	✓	✗			✓
*Existential Quantification	✗	✓	✗	$\sim$	$\sim$	✓
*Universal Quantification	✗	✓	✗			✓
*Minimum Unqualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Minimum Qualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Maximum Unqualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Maximum Qualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Exact Unqualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Exact Qualified Cardinality	✓	✓	✗	$\sim$	✓	✓
*Transitive Object Properties	✗	✓	✗	✗	✗	✓
Context-Sp. Exclusive OR of P.	✗	✓	✗		✓	✓
Context-Sp. Exclusive OR of P. Groups	✗	$\sim$	✗	✓	✓	✓
Allowed Values	✓	✓	✗	✓	✓	✓
Not Allowed Values	✗	✓	✗	✗	✓	✓
Literal Ranges	✗	✓	✗	✗	✗	✓
Negative Literal Ranges	✗	✓	✗	✗	✗	✓
Required Properties	✓	✓	✗	✓	✓	✓
Optional Properties	✓	✓	✗	✓	✓	✓
Repeatable Properties	✓	✓	✗	✓	✓	✓
Negative Property Constraints	✗	✓	✗		✓	✓
*Individual Equality	✗	✓	✗	✗	✗	✓
*Functional Properties	✗	✓	✗	✗	✗	✓
*Inverse-Functional Properties	✗	✓	✗	✗	✗	✓
*Value Restrictions		✓	✗			✓
*Self Restrictions	✗	✓	✗	✗	✗	✓
Primary Key Properties	✗	✓	✗	✗	✗	✓

Constraints Only Expressible by Other DSCLs.

constraint	DSP	OWL2-DL	OWL2-QL	ReSh	ShEx	SPIN
*Class-Specific Property Range	✓	✗	✗	✓	✓	✓
*Class-Sp. Reflexive Object P.	✗	✗	✗	✗	✗	✓
Membership in Controlled Vocabularies	✓	✗	✗	✗	✗	✓
IRI Pattern Matching	✗	✗	✗	✗	✓	✓
Literal Value Comparison	✗	✗	✗	✗	✓	✓
Define Order	✗	✗	✗	✗	✗	✓
Validation Levels	✗	✗	✗	✗	✗	✓
String Operations	✗	✗	✗	✗	✗	✓
Context-Specific Valid Classes	✗	✗	✗	✗	✗	✓
Context-Specific Valid Properties	✗	✗	✗	✗	✗	✓
Default Values	✗	✗	✗	✓	✗	✓
Mathematical Operations	✗	✗	✗	✗	✗	✓
Language Tag Matching	✗	✗	✗	✗	✗	✓
Language Tag Cardinality	✗	✗	✗	✗	✗	✓
Whitespace Handling	✗	✗	✗	✗	✗	✓
HTML Handling	✗	✗	✗	✗	✗	✓
Conditional Properties	✗	✗	✗	✗	✗	✓
Recommended Properties	✗	✗	✗	✗	✗	✓
Handle RDF Collections	✗	✗	✗	✗	✗	✓
Value is Valid for Datatype	✗	✗	✗	✗	✗	✓
*Exploiting Class/Property Specialization Ontology Axioms	✗	✗	✗	✗	✗	✓
*Cardinality Shortcuts	✗	✗	✗	✓	✓	✓

5 Computational Complexity of RDF Constraints

According to our evaluation of requirements to formulate RDF constraints, we identified three main ways of validating constraints.

1. validation by query answering without reasoning
2. validation by query answering with OWL 2 QL reasoning
3. validation by query answering with OWL 2 DL reasoning

The next table gives an overview of the computational complexity for RDF validation when it is performed -with and without reasoning.

Validation Type	Complexity
validation by query answering without reasoning	PSPACE-Complete
validation by query answering with OWL 2 QL reasoning	PTIME
validation by query answering with OWL 2 DL reasoning	N2EXPTIME

What is considered by rdf validation w/o reasoning, corresponds to performing SPARQL queries. It is known that performing SPARQL queries is in PSPACE-Complete [5]. Since OWL 2 profiles are based on DL-Lite family and query answering in OWL 2 QL is in LOGSPACE (or rather in AC^0) [3], so is the constraint validation by queries with reasoning. Furthermore, TBox reasoning in OWL 2 QL is in PTIME [3], hence complete query rewriting as well as combined complexity (reasoning and than querying) is in PTIME [1, 3]. With the more expressive profile OWL 2 DL, reasoning is in N2EXPTIME [4] which is a class of considerably higher complexity.

6 Related Work

7 Conclusion and Future Work

References

1. Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The dl-lite family and relations. *J. Artif. Int. Res.*, 36(1):1–69, September 2009.
2. Thomas Bosch, Andreas Nolle, Erman Acar, and Kai Eckert. Rdf validation requirements. Technical report, 2015.
3. Diego Calvanese, Giuseppe Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. Autom. Reason.*, 39(3):385–429, October 2007.
4. Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. Owl 2 web ontology language: Profiles. Technical report, December 2008.
5. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of sparql. *ACM Trans. Database Syst.*, 34(3):16:1–16:45, September 2009.