Using Description Logics for RDF Constraint Checking and Closed-World Recognition

Peter F. Patel-Schneider
Nuance Communications
1198 E. Arques Ave., Sunnyvale, California, U. S A.
pfpschneider@gmail.com

Abstract

RDF and Description Logics work in a open world setting where absence of information is not information about absence. Nevertheless, Description Logic syntax and semantics can be used for both constraint checking and closed-world recognition. When information is specified in the form of RDF graphs this constraint checking and closed-world recognition is simple to describe and can be efficiently performed in many cases.

There has recently been considerable attention paid to the problem of validating RDF or RDFS (Cyganiak, Wood, and Lanthaler 2014) information. There are several commercial systems that provide facilities for RDF validation, including TopQuadrant's SPIN (TopQuadrant 2011) and Clark&Parsia's Stardog ICV (Clark&Parsia 2014). There are several proposals for specifying the desired form of RDF information, such as Resource Shapes (Ryman 2014). In 2013 W3C held an RDF Validation Workshop (W3C 2013) to gauge interest in the area, and in the summer of 2014 W3C is in the process of starting a new working group on RDF validation (W3C 2014).

Just what, however, is RDF validation?

Some accounts and systems (such as Stardog ICV) identify RDF validation with satisfying integrity constraints, similar to checking database integrity constraints. In this account, there are conditions (constraints) placed on instance of classes, such as requiring that every person has a name and an address, both strings. The defining characteristic here is that explicit information is needed to satisfy the integrity constraint. To pass the constraint that a person has a name it is necessary to provide a particular string for the name of the person, and not just state that the person has some unknown name.

Other accounts and proposals (such as ShEx (Prud'hommeaux 2014; Solbrig and Prud'hommeaux 2014)) identify RDF validation more with recognition, similar to determining whether an individual belongs to a Description Logic (Baader et al. 2010) description. For example one might say that a person shape must have a name and an address and then ask which individuals satisfy the person shape constraint. Here, in contrast to the previous situation, the validation is divorced from any type information in the data. Again, however, there is the requirement that explicit information is needed to match the shape—a particular name must be provided, not just information that there must be one.

As shown in this paper, Description Logics can be used to provide the necessary framework for both checking constraints and providing closed-world recognition facilities, and thus cover what ShEx is trying to do and more, as evidenced by the considerable work on using Description Logics for constraints. (See the Related Work section.) Why then are there claims (Ryman, Hors, and Speicher 2013; Fokoue and Ryman 2013) that OWL (Motik, Patel-Schneider, and Parsia 2012)—the Semantic Web Description Logic—is inadequate for these purposes? There are several aspects of standard Description Logics that might not be consonant with constraints and the kind of recognition that might be desired. However, Description Logic syntax and semantics, and their instantiation in OWL, can serve as the basis for RDF constraint checking and closed-world recognition. The only change required is to consider a closed-world variation of the Description Logic semantics and then the development of RDF constraint checking and closed-world recognition is easy.

The Basic Idea

Closed-World Recognition

Let's first look at recognition. In recognition we want to determine whether a particular node in an RDF graph matches some criteria. For example, John in the RDF graph[1]

```
<#John> foaf:name "John"^^xsd:string .                    (1)
<#John> foaf:phone "+19085551212"^^xsd:string .
<#John> ex:friend <#Bill> .
<#John> ex:friend <#Willy> .
```

matches the ShEx shape

```
{ foaf:name xsd:string,                                   (2)
  foaf:phone xsd:string,
  ex:friend [2] }
```

because John has a string value for his name, a string value for his phone, and two friends.

Submitted to AAAI-2015.

[1] Throughout this paper Turtle (Prud'hommeaux and Carothers 2014) will be used for writing RDF graphs. Prefix and base statements will generally be omitted.

Determining whether an individual belongs to a Description Logic concept is also recognition. The Description Logic description[2] corresponding to the ShEx Shape (2) is

$$= 1\ \textit{foaf:name} \sqcap \forall \textit{foaf:name.xsd:string} \sqcap \quad (3)$$
$$= 1\ \textit{foaf:phone} \sqcap \forall \textit{foaf:phone.xsd:string} \sqcap$$
$$= 2\ \textit{ex:friend}$$

So it seems that Description Logics can easily handle ShEx recognition. Although the ShEx syntax is somewhat more compact here, as ShEx has constructs that combine number restrictions and value restrictions, the Description Logic syntax is not verbose and is quite reasonable.

However, John does not match (3) in a standard description logic. This is precisely because standard Description Logics, *and RDF*, do not assume that absence of information is information about absence. In a standard Description Logic, and also in RDF, John could have more than one name as far as the information in the above RDF graph is concerned. Many Description Logics (and, again, RDF too) also do not assume that different names refer to different individuals. So Bill and Willy could be the same person as far as the information in the above RDF graph is concerned.

It turns out that expressive Description Logics have facilities to explicitly state information about absence and information about differences and thus can be used to state complete information, at least on a local level. For example, if we add information to (1) stating that John has only one name and phone, that John's only friends are Bill and Willy, *and* that Bill is not the same as Willy, as in

$$\textit{<\#John>} \in \leq 1\ \textit{foaf:name}$$
$$\textit{<\#John>} \in \leq 1\ \textit{foaf:phone}$$
$$\textit{<\#John>} \in \forall \textit{ex:friend}.\{\textit{<\#Bill>}, \textit{<\#Willy>}\}$$
$$\textit{<\#Bill>} \neq \textit{<\#Willy>}$$

then John does match (3).

So it is not that Description Logics (including OWL) do not perform recognition as in ShEx, it is just that Description Logics do not make the assumption that absence of information is information about absence. In expressive Description Logics (again including OWL) it is possible to explicitly state what comes implicitly from the assumption that absence of information is information about absence.

However, suppose that we want to make this assumption generally? We could manually add a lot of axioms like the ones above, but this is both tedious and error-prone, and thus not at all a viable solution. Instead we can proceed by making the assumption that if the truth of some fact cannot be determined from the information given, then that fact is false. This is often called the closed world assumption or negation by failure, as the failure to prove some fact is used to support its falsity. There is a very large body of work on this topic and there are many tricky questions that arise with respect to closure in any sophisticated formalism, and expressive Description Logics (including OWL) are indeed sophisticated. As well, reasoning in expressive Description Logics that also have closed world facilities is extremely difficult, even in simple cases.

Fortunately RDF and RDFS are unsophisticated and inexpressive, so neither the tricky questions nor the reasoning difficulties arise if all information comes in the form of RDF triples under the RDF or RDFS semantics (Hayes and Patel-Schneider 2014). The basic idea is to treat the triples (and their RDF or RDFS consequences, if desired) as completely describing the world. So, 1/ if a triple is not present then it is false and 2/ different IRIs describe different individuals. This is precisely the same idea that underlies model checking, where a model is a finite set of ground first-order facts and everything else is false. First-order inference is undecidable, but determining whether a first-order sentence is true in one particular model is much, much easier.

In this way it is possible to use the Description Logic syntactic and semantic machinery to define how to recognize descriptions under the same assumptions that underlie ShEx. The only change from the standard Description Logic setup is to define how to go from an RDF graph to the Description Logic model that the RDF graph embodies. Definitions, even recursive definitions, can also be handled.

This is all quite easy and conforms to a common thread of both theoretical and practical work. It also matches the theoretical underpinning of Stardog ICV (Clark&Parsia 2014). Further, the approach can be implemented by translation into SPARQL queries, showing that it is practical. (There may be some constructs of very expressive description logics that do not translate into SPARQL queries when working with complete information, but at least the Description Logic constructs that correspond to the usual recognition conditions do so translate.)

Constraint Checking

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

Inference is the process of determining what follows from what has been stated. Inference ranges from simple (John is a student, students are people, therefore John is a person) to the very complex. Determining whether a constraint holds is just determining whether the constraint can be inferred from the given information.

Again, however, constraint checking is generally done with respect to complete information. So, to determine whether the constraint

$$\begin{aligned}\textit{<\#John>} \in \textit{ex:Person} \sqcap \quad (4)\\ = 1\ \textit{foaf:name} \sqcap \forall \textit{foaf:name.xsd:string} \sqcap\\ = 1\ \textit{foaf:phone} \sqcap \forall \textit{foaf:phone.xsd:string} \sqcap\\ = 2\ \textit{ex:friend}\end{aligned}$$

is valid in the presence of (locally) complete information such as

$$\textit{<\#John>} \in \textit{ex:Person}$$
$$\textit{<\#John>}\ \textit{foaf:name}\ \text{"John"}\hat{}\hat{}\textit{xsd:string}\ .$$
$$\textit{<\#John>} \in = 1\ \textit{foaf:name}$$

[2]The abstract syntax (Baader et al. 2010) for Description Logics—a compact but non-ASCII syntax—will be used throughout this paper.

Figure 1: Data for Example

<#*Amy*> *rdf:type ex:UniStudent* .	<#*Amy*> *ex:enrolled* <#*SUNYOrange*> .	<#*Amy*> *ex:friend* <#*Bill*> .
<#*Amy*> *foaf:name* "Amy"^^*xsd:string* .	<#*Bill*> *ex:enrolled* <#*ReindeerPoly*> .	<#*Amy*> *ex:friend* <#*John*> .
<#*Bill*> *rdf:type ex:UniStudent* .	<#*Bill*> *ex:enrolled* <#*HudsonValley*> .	<#*Bill*> *ex:friend* <#*Amy*> .
<#*Bill*> *foaf:name* "Bill"^^*xsd:string* .	<#*John*> *ex:enrolled* <#*ReindeerPoly*> .	<#*Bill*> *ex:friend* <#*John*> .
<#*John*> *rdf:type ex:GradStudent* .	<#*Susan*> *ex:enrolled* <#*ReindeerPoly*> .	<#*John*> *ex:friend* <#*Amy*> .
<#*John*> *foaf:name* "John"^^*xsd:string* .	<#*Susan*> *ex:enrolled* <#*SUNYOrange*> .	<#*John*> *ex:friend* <#*Bill*> .
<#*John*> *ex:supervisor* <#*Len*> .	<#*Susan*> *ex:enrolled* <#*HudsonValley*> .	<#*John*> *ex:friend* <#*Len*> .
<#*Susan*> *rdf:type ex:Person* .	<#*Len*> *ex:affiliation* <#*ReindeerPoly*> .	<#*Len*> *ex:friend* <#*Amy*> .
<#*Susan*> *foaf:name* "Susan"^^*xsd:string* .	<#*Len*> *ex:affiliation* <#*SUNYOrange*> .	<#*Len*> *ex:friend* <#*Susan*> .
<#*Len*> *rdf:type ex:Faculty* .	<#*SUNYOrange*> *rdf:type ex:ResOrg* .	
<#*Len*> *foaf:name* "Len"^^*xsd:string* .	<#*HudsonValley*> *rdf:type ex:Uni* .	

Figure 2: RDFS Ontology for Example

foaf:name rdfs:range xsd:string .	*ex:enrolled rdfs:domain ex:UniStudent* .
ex:UniStudent rdfs:subClassOf ex:Person .	*ex:enrolled rdfs:range ex:Uni* .
ex:GradStudent rdfs:subClassOf ex:UniStudent .	*ex:supervisor rdfs:domain ex:GradStudent* .
ex:Faculty rdfs:subClassOf ex:Person .	*ex:supervisor rdfs:range ex:Faculty* .
ex:Uni rdfs:subClassOf ex:Organization .	*ex:affiliation rdfs:domain ex:Person* .
ex:ResOrg rdfs:subClassOf ex:Organization .	*ex:affiliation rdfs:range ex:Organization* .

Figure 3: Constraints and Recognition Axioms for Example

1	$ex{:}Person \sqcap ex{:}Organization \equiv \{\}$	7 $ex{:}Faculty \sqsubseteq \leq 5\ ex{:}supervisor^{-}.ex{:}GradStudent$
2	$ex{:}Person \sqsubseteq = 1\ foaf{:}Name \sqcap \forall foaf{:}Name.xsd{:}string$	8 $ex{:}Uni \sqsubseteq \geq 2\ ex{:}enrolled^{-}$
3	$ex{:}UniStudent \sqsubseteq \geq 1\ ex{:}enrolled \sqcap \forall ex{:}enrolled.ex{:}Uni$	9 $ex{:}GradStudent \mid ex{:}enrolled \sqsubseteq ex{:}supervisor \circ ex{:}affiliation$
4	$ex{:}GradStudent \sqsubseteq = 1\ ex{:}enrolled \sqcap \forall ex{:}enrolled.ex{:}ResOrg$	<#*HecticStudent*> $\equiv \geq 3\ ex{:}enrolled$
5	$ex{:}Faculty \sqsubseteq \geq 1\ ex{:}affiliation \sqcap \forall ex{:}affiliation.ex{:}Uni$	<#*StudentFriend*> $\equiv \geq 2\ ex{:}friend.$<#*StudentFriend*>
6	$ex{:}Faculty \sqsubseteq \leq 1\ ex{:}affiliation.ex{:}ResOrg$	

<#*John*> *foaf:phone* "+19085551212"^^*xsd:string* .
<#*John*> $\in = 1$ *foaf:phone*
<#*John*> *ex:friend* <#*Bill*> .
<#*John*> *ex:friend* <#*Willy*> .
<#*John*> *ex:friend* <#*Susan*> .
<#*John*> $\in \forall ex{:}friend.\{$<#*Bill*>, <#*Willy*>, <#*Susan*>$\}$
<#*Bill*> $\neq$ <#*Willy*>
<#*Bill*> $\neq$ <#*Susan*>
<#*Susan*> $\neq$ <#*Willy*>

is simply a matter of determining whether the constraint follows from the information. (Generally constraints like (4) are written to handle all the members of a class as in

$$ex{:}Person \sqsubseteq = 1\ foaf{:}name \sqcap \forall foaf{:}name.xsd{:}string \sqcap = 1\ foaf{:}phone \sqcap \forall foaf{:}phone.xsd{:}string \sqcap = 2\ ex{:}friend$$

instead of just a single node, but the principle is the same.)

So a way to do constraints in Description Logics is to first set up complete information, and then just perform inference. This approach has been explored in the context of letting certain roles be completely specified as in a database (Patel-Schneider and Franconi 2012). Other approaches to constraints in Description Logics (de Bruijn et al. 2005; Motik, Horrocks, and Sattler 2009; Tao et al. 2010; Donini, Nardi, and Rosati 2002; Sengupta, Krisnadhi, and Hitzler 2011) are considerably more complex, as they deal with the complexities that arise when there are multiple ways to complete the information. However, all these approaches largely agree when there is only a single way to complete the information.

Setting up complete information is just what was done above for closed-world recognition, so this technique can also be used for constraint checking. Of course, this doesn't mean that you have to implement Description Logic inference with complete information the same way that you need to with incomplete information. In fact, as above, constraint checking can be implemented as SPARQL queries.

Example

Here is a small example of how Description Logic constructs can be used for constraint checking.

There are three separate kinds of information. The RDF triples in Figure 1 provide the data for the example. The RDFS ontology in Figure 2 provides the organization of the data. The Description Logic axioms in Figure 3 provide the constraints to be validated against the data and the ontology and the classes vocabulary for closed-world recognition.

The constraints are all satisfied, except for one, as follows:

1. Nothing can be inferred to be both a person and an organization, and so persons and organizations are disjoint.

2. Every object that can be inferred to be a person (i.e., students and faculty) has a single name provided, and that name is a string, so every person has exactly one name in the closure.
3. All students (and grad students) are enrolled in universities—the range of *ex:enrolled* is *ex:Uni*, which makes the typing part of the constraint redundant here.
4. Reindeer Poly is not specified to be a research organization, so although John is enrolled exactly once the constraint on graduate students being enrolled in research organizations is not satisfied.
5. All faculty (namely Len) are affiliated with only universities.
6. All faculty (again only Len) are affiliated with at most one research organization, as Reindeer Poly is not specified to be a research organization and Len is only affiliated with SUNY Orange and Reindeer Poly.
7. All faculty supervise fewer than five graduate students, as the only faculty (Len) only supervises one student (John).
8. Each university (SUNY Orange, Reindeer Poly, and Hudson Valley) has at least two students enrolled in it, because Amy, Bill, John, and Susan are different individuals.
9. For every graduate student enrollment (*ex:enrolled* domain-restricted to *ex:GradStudent*) there is a supervisor of the graduate student affiliated with the university.

The only hectic student is Susan, as she is the only person with at least three enrollments. Amy, Bill, and John all belong to <#*StudentFriend*> because when maximally interpreting <#*StudentFriend*> they each have at least two friends who belong to <#*StudentFriend*>. Len does not belong to <#*StudentFriend*> even though he has two friends, because Susan has too few friends and cannot belong to <#*StudentFriend*>.

One might want to validate that domain and range types are not inferred, but are instead explicitly stated in the data. This can be done by using a version of the ontology without the domain and range statements and validating against a set of constraints that just have the removed domain and range statements. In the example, this would detect that Susan was not stated to be a student, violating the domain constraint for *ex:enrolled*; that SUNY Orange and Reindeer Poly were not stated to be universities, violating the range constraint for *ex:enrolled*; and that Reindeer Poly was not stated to be an organization, violating the range constraint for *ex:affiliation*. All other domain and range constraints would be satisfied, as some required class memberships would be inferred from the subclass statements.

Related Work

The closest work in a technical sense is the work of Patel-Schneider and Franconi (2012). In that work some properties and classes were considered as closed, which turned description logic axioms involving those properties and classes into constraints. RDF and RDFS are very similar to a situation where all properties and classes are closed. The current paper adds the idea of closed-world recognition, which was only implicit in the previous work, and maximal extensions, which provide a much better treatment of recursive definitions, particularly in the monotone case.

The work of Motik, Horrocks, and Sattler (2009) and of Tao *et al.* (2010) are similar in that they both divide up axioms into regular axioms and constraints. They both also permit general Description Logic axioms, not just RDF or RDFS graphs as here. To handle full Description Logic information requires a much more complex construction, involving minimal interpretations. They also do not consider closed-world recognition.

The work of Sengupta, Krisnadhi, and Hitzler (2011) uses circumscription as the mechanism to minimize interpretations. It is otherwise similar to the previous efforts. The work of Donini, Nardi, and Rosati (2002) uses autoepistemic constructs within axioms to model constraints, and is thus quite different from the approach here.

OWL Flight (de Bruijn et al. 2005) is a subset of OWL where axioms are given meaning as Datalog constraints. Again, as an expressive Description Logic is handled the construction is more complex than the one here.

The Details, but Not All the Details

Description Logic Semantics

The semantics of Description Logics are generally given as a model theory, as for OWL DL (Motik, Patel-Schneider, and Parsia 2012). OWL DL has a complex semantics, as far as Description Logics go, to cover all its constructs and to make it more compatible with RDF. The semantics here will follow the semantics of OWL, with the exception that any property can have both individuals, e.g., <#*John*>, and data values, e.g., "John"^^*xsd:string*, as values.

The basics of Description Logics semantics are interpretations, which provide a meaning for the primitive constructs in terms of a particular domain of discourse. The meaning of an individual name, such as <#*John*>, is an element of this domain, here that element that we think of as being John. Literal values, such as "John"^^*xsd:string*, are treated specially—their meaning is determined by their datatype. The meaning (or interpretation) of a named concept such as *ex:person*, is a subset of the domain, here those individuals that we might think of as people. The meaning of a named property, such as *ex:friend*, is a set of pairs over the domain, here those pairs that we might think of as being the friend-of relationship. The meaning of non-primitive constructs, such as the description $= 2$ *ex:friend*, are built up from these primitives, resulting here in the set of domain elements that are related to exactly two domain elements via the meaning of *ex:friend*.

Axioms, such as <#*John*> $\in$ *ex:Person*, are true precisely when the meaning of their parts satisfies a particular relationship, here that the meaning of <#*John*> is an element of the meaning of *ex:Person*. There are some other aspects to this simple story, to handle the differences between individuals, e.g., <#*John*>, and data values, e.g., "John"^^*xsd:string*, and to make reasoning over some constructs easier. A Description Logic model of a set of axioms (including what we

might call facts), is then just an interpretation that makes all the axioms true.

Canonical Interpretations of RDF Graphs

In an interpretation everything is specified, so each interpretation has complete information. The basic idea is thus to construct an interpretation making just the triples in an RDF graph true and then work with that single interpretation. In this way information that is absent from the RDF graph is considered false.

We can think of most RDF graphs as sets of Description Logic axioms, particularly as we are ignoring the common Description Logic division of properties into properties that have objects that are individuals and properties that have objects that are data values.[3] This correspondence breaks down in two areas: 1/ when the built-in RDF and RDFS vocabulary is used in unusual ways (e.g., making *rdfs:subClassOf* a sub-property of *rdfs:subPropertyOf*), and 2/ if reasoning about individuals can affect reasoning about classes (e.g., forcing two individuals that are also classes to be the same). The abuse and extension of the built-in RDF and RDFS vocabulary is rare, so we just exclude these RDF graphs from our account. (Particular extensions of the RDF and RDFS vocabulary could be built in to an extension of the approach given here.) In RDF and RDFS, but not in OWL, reasoning about individuals cannot affect reasoning about classes are forced to belong to finite datatypes. This is also rare, so we will not handle inferences following from these situations.

Given an RDF graph, we construct the canonical Description Logic interpretation of the graph as follows.

1. The domain of the interpretation consists of the non-literal nodes of the RDF graph plus the properties in the graph and the mapping for nodes is the identity mapping. (One might think that this is not an appropriate way to construct an interpretation, as it sets the meaning of <#*John*> to <#*John*>, not anything that we might think of as being John, but as far as the formal machinery is concerned, the actual domain elements are not important.)
2. Named classes are formed for each node in the graph that has an *rdf:type* link with it as an object or belongs to *rdfs:Class* and their extensions are the set of nodes for which *rdf:type* triples link them to the class.
3. Named properties (remember we are ignoring the usual Description Logic division of properties) are formed for each predicate in the graph and each node that belongs to *rdf:Property*, and their extensions are the set of pairs that correspond to triples in the graph with the property as predicate.
4. Datatypes are formed for all the datatypes in the graph, and given meaning in the usual way.

So from the initial RDF graph in this paper, we end up with a canonical interpretation with six domain elements, <#*John*>, <#*Bill*>, <#*Willy*>, *foaf:name*, *foaf:phone*, and *ex:friend*. The interpretation of *foaf:name* consists of just $\langle$<#*John*>, "*John*"$\rangle$ The interpretation of *foaf:phone* consists of just $\langle$<#*John*>, "*+19085551212*"$\rangle$. The interpretation of *ex:friend* consists of $\langle$<#*John*>, <#*Bill*>$\rangle$ and $\langle$<#*John*>, <#*Willy*>$\rangle$.

For constraints and descriptions that use only vocabulary in the RDF graph all we do is work with this interpretation and consider whether the constraint axiom is true in this interpretation so the development is easy. It is obvious that <#*John*> belongs to the interpretation of the first Description Logic description given above, as expected. Each ShEx shape construct, except for the constructs that match predicate IRIs by pattern, can be converted into a Description Logic description that exactly corresponds to the shape.[4]

Extending to New Classes

For closed-world recognition, it is useful to define new classes, as in

<#*PurePerson*> $\equiv \geq 1$ *ex:friend* $\sqcap$
$\forall$*ex:friend*.<#*PurePerson*>

There are several possibilities for the meaning of recursively-defined new classes. The new classes could be interpreted as broadly as possible, as narrowly as possible, or in any consistent manner.

It appears in ShEx that such classes as to be interpreted as broadly as possible. For example, in the RDF graph

<#*John*> *ex:friend* <#*Bill*> .
<#*Bill*> *ex:friend* <#*John*> .

the ShEx approach would be that both <#*John*> and <#*Bill*> belong to <#*PurePerson*>. We will take this approach here and interpret new classes as broadly as possible

New classes are handled by considering extensions of the interpretations defined as above. An extension of an interpretation is a new interpretation 1/ with the same domain as the original interpretation, and 2/ that has the same meaning for all individuals, named classes, and named properties in the original interpretation. The extension is allowed to have new named classes, but not new named properties or new individuals. New individuals are not allowed because some Description Logic constructs are sensitive to the set of individuals. New properties are not allowed because they may increase the computational complexity of closed-world recognition.

An interpretation is an extended canonical model of an RDF graph with respect to a set of constraints if it is an extension of the canonical model of the RDF graph and is a model of the constraints.

To interpret recursively defined classes as broadly as possible not all extended canonical models are considered, only maximal ones. An model is maximal among a set of models if there is no other model in the set that 1/ interprets all classes as supersets of their interpretation in the maximal model, and 2/ interprets at least one class as a strict superset of its interpretation in the maximal model.

[3] This division has been made so that reasoners do not have to worry about data values having properties, but if we are constructing models this is not a problem. It is easy to revise the treatment here to bring back this division.

[4] The demonstration of this claim is in the supplementary material for this paper.

An individual is recognized as belonging to a description if its interpretation belongs to the interpretation of the description in all maximal extended canonical models.

It turns out that there is only one (up to isomorphism) maximal extended canonical model of the above definition of <#*PurePerson*>, and both <#*John*> and <#*Bill*> are in the extension of <#*PurePerson*> in this model. If all new classes are monotone in all the other new classes (i.e., if the extension of some class grows then no other class extensions shrink) then there is always exactly one maximal extension and it corresponds to the apparent meaning of recursive shapes in ShEx.[5]

RDF and RDFS Semantics

So everything looks fine. We go from an RDF graph to a slightly modified Description Logic interpretation and from there perform constraint checking by determining whether a set of Description Logic axioms are satisfied in the interpretation or in a set of maximal extensions of the interpretation. We can also perform closed-world recognition by determining the interpretation of the new defined classes in the axioms and these classes can even be defined recursively.

However, there is one missing part of the story. If the RDF graph includes triples that trigger RDF or RDF inferences that are not already in the RDF graph the interpretation will not look like an RDF (or RDFS) interpretation. For example, if the RDF graph is

<#*John*> *rdf:type ex:Student* .
ex:Student rdfs:subClassOf ex:Person .

our canonical interpretation states that <#*John*> does not belong to *ex:Person*, which goes against the RDFS meaning of the above graph.

Fortunately, it is relatively easy to recover from this problem. All that is needed is to add all the RDF (or RDFS) consequences to the graph. Yes, there are an infinite number of these consequences, but our formal development does not care whether the graph is finite or infinite. For complexity analysis and implementation it is not hard to come up with an finite representation of these consequences, like the one initially done by ter Horst (2005), and make the minor fixups needed to determine correct answers from the answers gleaned from this finite approximation.

This all works because the RDF (or RDFS) consequences of an RDF graph can be represented as an RDF graph. OWL consequences cannot be so represented, as the consequences in OWL can be disjunctive. This requires working with minimal equality and minimal models or some other way to single out only the desired interpretations as in previous work on Description Logic constraints, making the formal development much harder and presenting many more choices that have to be justified.

Complexity

Checking constraints and performing closed-world recognition is not difficult, as long as any recursive loops are monotone. It is not hard to see that checking axioms against an interpretation is polynomial, as long as there is no new vocabulary in the axioms.

Maximal extensions can be computed in the monotone setting by a recursive process that when it encounters the same class on the same individual again tentatively assumes that the individual belongs to the class. If nothing counters any of these assumptions then they all are considered to be true, essentially expanding all the recursive classes all at once. Other techniques, such as the ones used in Datalog can also be used. As the number of individuals considered need only be slightly more than the size of the RDF graph, again using the work of ter Horst (2005), this process is also polynomial.

These techniques cannot be used for non-monotone recursive definitions, as expanding one class or property might reduce another.

Implementation

The work of Tao *et al.* (2010) shows that the standard Description Logic constraints can be partly implemented as SPARQL queries when no new vocabulary is used. Tao et al. worked in a general OWL setting, where their approach is sound but not complete, but in an RDF setting the approach is both sound and complete, because there is only a single model that needs to be considered. Implementation experience with Stardog ICV (Clark&Parsia 2014) shows that this implementation method is effective in practice. (Recent work at Mannheim implements OWL descriptions as constraints using a similar approach.)

Non-recursive closed-world recognition can be handled by using nested or repeated SPARQL queries. Monotone recursive closed-world recognition can be implemented using Datalog techniques. Non-monotone recursive closed-world recognition is more complex and cannot be handled in the same way. This indicates that excluding non-monotone recursive closed-world recognition could be a reasonable stance to take.

Conclusion

Description Logics can indeed be used for both the syntax and semantics of constraint checking and closed-world recognition in RDF, by employing an analogue of model checking, and much of both constraint checking and closed-world recognition can be effectively implemented using a translation to SPARQL queries. The main difference between closed-world recognition and constraint checking is that the former either has no axioms or only uses axioms defining names that do not occur in the RDF graph whereas constraint checking uses axioms that relate concepts appearing in the RDF graph to descriptions.

By restricting the knowledge to be RDF or RDFS we obtain a simpler formulation, easy implementation, and good performance as compared to previous work in this area. The approach here can be easily extended to other subsets of OWL that have a unique minimal model. An OWL profile with this property is OWL RL (Motik et al. 2012).

[5]Proof sketches of claims here and later in the paper are in the supplementary material for this paper.

References

Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D.; and Patel-Schneider, P. F., eds. 2010. *The Description Logic Handbook: Theory, implementation, and applications*. Cambridge University Press, 2nd edition.

Clark&Parsia. 2014. Stardog integrity constraint validation. *http://docs.stardog.com/icv*.

Cyganiak, R.; Wood, D.; and Lanthaler, M. 2014. RDF 1.1 concepts and abstract syntax. W3C Recommendation, *http://www.w3.org/TR/rdf-concepts/*.

de Bruijn, J.; Polleres, A.; Lara, R.; and Fensel, D. 2005. OWL DL vs. OWL Flight: Conceptual modeling and reasoning for the semantic web. In *Proceedings of the Fourteenth World Wide Web Conference*.

Donini, F. M.; Nardi, D.; and Rosati, R. 2002. Description logics of minimal knowledge and negation as failure. *ACM Transactions on Computational Logic* 3(2):177—225.

Fokoue, A., and Ryman, A. 2013. OSLC resource shape: A linked data constraint language. *www.w3.org/2001/sw/wiki/images/b/b7/RDFVal_Fokuoe_Ryman.pdf*.

Hayes, P., and Patel-Schneider, P. F. 2014. RDF 1.1 semantics. W3C Recommendation, *http://www.w3.org/TR/rdf11-mt/*.

Motik, B.; Grau, B. C.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2012. OWL 2 Web Ontology Language: Profiles (second edition). W3C Recommendation, *http://www.w3.org/TR/owl2-profiles*.

Motik, B.; Horrocks, I.; and Sattler, U. 2009. Bridging the gap between OWL and relational databases. *Journal of Web Semantics* 7(2):74–119.

Motik, B.; Patel-Schneider, P. F.; and Parsia, B. 2012. OWL 2 web ontology language: Structural specification and functional-style syntax. W3C Recommendation, *http://www.w3.org/TR/owl2-syntax/*.

Patel-Schneider, P. F., and Franconi, E. 2012. Ontology constraints in incomplete and complete data. In *Proceedings of the Eleventh International Semantic Web Conference*. Springer.

Prud'hommeaux, E., and Carothers, G. 2014. RDF 1.1 turtle: Terse RDF triple language. W3C Recommendation, *http://www.w3.org/TR/turtle/*.

Prud'hommeaux, E. 2014. Shape expressions 1.0 primer. W3C Member Submission, *http://www.w3.org/Submission/shex-primer/*.

Ryman, A. G.; Hors, A. J. L.; and Speicher, S. 2013. OSLC resource shape: A language for defining constraints on linked data. In Bizer, C.; Heath, T.; Berners-Lee, T.; Hausenblas, M.; and Auer, S., eds., *Proceedings of the WWW2013 Workshop on Linked Data on the Web (LDOW2013)*. *ceur-ws.org/Vol-996/papers/ldow2013-paper-02.pdf*.

Ryman, A. 2014. Resource shape 2.0. W3C Member Submission, *http://www.w3.org/Submission/shapes/*.

Sengupta, K.; Krisnadhi, A.; and Hitzler, P. 2011. Local closed world semantics: Grounded circumscription for OWL. In *Proceedings of the Tenth International Semantic Web Conference*, 617–632. Springer.

Solbrig, H., and Prud'hommeaux, E. 2014. Shape expressions 1.0 definition. W3C Member Submission, *http://www.w3.org/Submission/shex-defn/*.

Tao, J.; Sirin, E.; Bao, J.; and McGuinness, D. L. 2010. Integrity constraints in OWL. In *Proceedings of the Twenty-Fourth National Conference on Artificial Intelligence*. American Association for Artificial Intelligence.

ter Horst, H. J. 2005. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics* 3(2-3):79–115.

TopQuadrant. 2011. SPIN—modeling vocabulary. W3C Member Submssion, *http://www.w3.org/Submission/spin-modeling/*.

W3C. 2013. W3C RDF validation workshop report. *http://www.w3.org/2012/12/rdf-val/*.

W3C. 2014. W3C RDF data shapes charter. *http://www.w3.org/2014/data-shapes/charter*.