Towards a Scalable, Pragmatic Knowledge Representation Language for the Web

Florian Fischer, Gulay Unel, Barry Bishop, and Dieter Fensel

Semantic Technology Institute (STI) Innsbruck,
University of Innsbruck, Austria
firstname.lastname@sti2.at

Abstract. A basic cornerstone of the Semantic Web are formal languages for describing resources in a clear and unambiguous way. Logical underpinnings facilitate automated reasoning about distributed knowledge on the Web and thus make it possible to derive only implicitly available information.

Much research is geared to advancing very expressive formalisms that add increasingly complex modelling constructs. However, this increase in language expressivity is often intrinsically linked to higher computational cost and often leads to formalisms that have high theoretical complexity and that are difficult to implement efficiently.

In contrast, reasoning in the context of the Web has a distinct set of requirements, namely inference systems that can scale to planetary-size datasets. A reduced level of expressivity is often sufficient for many practical scenarios and crucially, absolutely necessary when reasoning with such massive datasets. These requirements have been acknowledged by active research towards more lightweight formalisms and also by industrial implementations that often implement only tractable subsets of existing standards.

In this paper we aim to explore this trend and formulate a basic language, called $L2$, layered upon RDF as the data-model, that is inherently tractable, easy to implement on common rule engines and motivated by pragmatic considerations concerning the use of language constructs and the means to implement them.

1 Introduction

The next evolutionary step for the Web, the Semantic Web [1], envisions human-readable content enriched with meta-data that has machine-understandable semantics for the purpose of sharing and interconnecting commercial, scientific, personal, and other data. Using a well defined formal language for this purpose enables machine interpretability and in turn automated processing. This vision leads to a Semantic Web, in which content has a well defined meaning and can be reasoned with in order to derive implicit knowledge.

The Web has made tremendous amounts of information available that can be processed based on the formal semantics attached to it, e.g. as a product

A. Pnueli, I. Virbitskaite, and A. Voronkov (Eds.): PSI 2009, LNCS 5947, pp. 124–134, 2010.

of the Linking Open Data (LOD)[1] [2] community. A number of languages have been developed that use logic for the purpose of defining these formal semantics. However, the initial sets of standards for this purpose, e.g. OWL [3], have very high worst-case complexity results for key inference problems (usually ExpTime or higher).

The inherent trade-off between the expressiveness of a logical representation formalism and scalability of reasoning has been clearly observed from a theoretical point of view [4] and has also been shown to have a very practical impact on possible use-cases. While worst-case complexity results might not always reflect the practical behavior of an implementation they become increasingly important when faced with the sheer size of the data that is involved in reasoning at a Web scale. Furthermore, data found on the Web is not only special in terms of size, but also in terms of diversity, and in turn inconsistency. Consequently, since completeness in the traditional sense might be a hopeless endeavor, it makes sense to focus only on a pragmatically selected subset of inferences that

- provide a useful level of additional semantics for end-users on the web, falling in line with language constructs that are actually employed,
- are inherently tractable in terms of computational complexity,
- can be practically implemented without major obstacles, or that are already supported by existing tools.

As a contribution towards this goal, we propose $L2$, a very lightweight formalism that supports tractable inferences by both omitting "expensive" language constructs and in certain cases "weakening" the semantics of them. $L2$'s intended semantics is defined as set of "entailment rules" that operate directly on RDF triples, and are thus independent of any particular high level syntax.

This paper is structured as following: Section 2 motivates our approach and describes related work. Section 3 outlines the features of $L2$. Section 4 extends this high-level view with the relevant formal underpinnings in the form of entailment rules that operate directly on a set of RDF triples, and specify $L2$'s intended semantics. Finally, Section 5 concludes and summarizes this paper.

2 Motivation and Related Work

RDF as a data-model represents a labeled, directed multi-graph. Layered upon this are more expressive languages such as RDF Schema [5] and OWL [3], which were introduced to provide a greater degree of expressive power. A fundamental result is that even small increases in the expressive power of a language can have a severe impact on the associated reasoning complexity that leads to the intractability of inference. However, as [6] and others point out, a large portion of Semantic Web data is often only described using a limited subset of existing standards, i.e. RDFS plus certain elements from OWL. Moreover, an important

[1] `http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`

observation is that while resources on the Web are likely to be annotated with relatively lightweight ontologies, the number of resources annotated with these ontologies is likely to be very large [7].

Practical computational efficiency is important and is reflected both by active research on tractable, lightweight formalisms such as DL-Lite [8], EL++ [9], pD* [10], ELP [11], ... as well as the adoption of tractable profiles within the upcoming OWL 2 standard [12].

Aside from the theoretical work in this area, it is notable that existing implementations of large-scale, RDF-based inference engines often support only a specific subset of current standards in order to scale to very large data-sets. These inference engines give an indication of what modeling primitives are useful and are usually a combination of primitives with low complexity overhead or are based on practical user requirements. In other words, language features are not considered purely in terms of their theoretical characteristics, but also in terms of:

1. The relevance of specific language constructs for users.
2. The practicability of implementing certain language features efficiently.

The subsets implemented, e.g. in OWLIM[2], Oracle 11g[3], or AllegroGraph RDFStore[4] all support a very similar set of language primitives, usually with the aim of avoiding inferences that derive disjunctions or the existence of anonymous individuals. The features supported usually include support for the basic features available in RDFS and additionally specific parts of the OWL vocabulary along the lines of [10]. More particularly, all of these products support (to various extents) class and property hierarchies, equivalence (of properties, classes and individuals), and additional qualitative statements about properties (denoting transitivity, symmetricity, etc).

3 Language Overview

In this section we describe the language primitives of $L2$, which are selected based on (i) practical considerations outlined in the previous section, (ii) theoretical complexity results. This selection mostly consists of the RDFS vocabulary and a limited sub-set of OWL that is still inherently tractable. As $L2$ could be considered an OWL fragment, in the sense that it allows a restricted set of inferences to be made, we can use the OWL 2 functional-style[5] as high-level syntax. However, any surface syntax with an appropriate mapping to the underlying RDF primitives can be used, e.g. [13], because the fundamental design aspects of the language are independent of the particular syntax employed. This achieves two goals: First of all, it automatically aligns $L2$ with existing standards and at the same time facilitates easy end-user adoption.

[2] http://www.ontotext.com/owlim/index.html
[3] http://www.oracle.com/technology/tech/semantic_technologies/index.html
[4] http://agraph.franz.com/allegrograph/
[5] http://www.w3.org/TR/owl2-syntax/

We will now briefly enumerate the features in $L2$, explain why they are included and provide informal descriptions.

Class definitions. (`rdfs:Class`) A class defines a set of individuals that belong together because they share common properties. Only partial class definitions are supported and not complete class definitions, because they allow the emulation of several other language features that are not explicitly included, e.g. class intersection.

Subclass descriptions. (`rdfs:subClassOf`) $L2$ allows the definition of class hierarchies in the same way as RDFS. Thus the intended meaning is exactly the same: If class C_1 is defined to be the subclass of a class C_2 then the set of individuals that "belong to" (are in the class extension of) C_1 should be a subset of those that belong to C_2. Furthermore, subclass relations are transitive and a class is a subclass of itself.

Property definitions. (`rdf:Property`) Properties can be used to state specific relations, either between individuals or between individuals and plain data values.

Subproperty descriptions. (`rdfs:subPropertyOf`) In the same fashion as for classes it is also possible to organize properties in hierarchies by stating that a property is a sub-property of a number of other properties. Obviously, as subclassing, `rdfs:subPropertyOf` has transitive behavior.

Domain and Range restrictions. (`rdfs:domain` and `rdfs:range`) The domain of a property restricts what individuals the property can be applied to, while the range restricts the set of values that a property can take. Both domain and range restrictions impose *global* constraints on a property independently of which specific class a property is applied to. It needs to be noted that for both domains and ranges, it is possible to give two different kinds of interpretations, namely inferring and constraining. For example, assume an individual x that is related to another individual y via a certain property p, with a class C_1 as domain and another class C_2 as the range. Applying an inferring interpretation, it is possible to conclude that x belongs to C_1 and furthermore that y belongs to C_2. A constraining interpretation on the other hand, would actually *check* that the individual is of the correct type, as a condition, and otherwise raise this as an error. Both semantics of domain and ranges are valid and a choice should be made depending on the requirements of an application.

Class equivalence. (`owl:equivalentClass`) Two classes may be stated to be equivalent, in which case they also have the same set of instances and moreover also share common super and subclasses. This functionality is useful to perform basic schema mapping. Class equivalence is a symmetric, reflexive, and transitive property. Furthermore, class equivalence between two classes C_1 and C_2 simply requires two implications stating that C_1 is a subclass of C_2 and vice versa. In this sense it is cleanly layered on top of RDFS, where this functionality is already available, but with no explicit syntax.

Transitive properties. (`owl:TransitiveProperty`) Transitivity of properties has the usual meaning that if a property p holds for a pair of individuals (x, y) and another pair (y, z), then it also holds for (x, z).

Symmetric properties. (`owl:SymmetricProperty`) A symmetric property is a property that is true in both directions. $L2$ allows for the specification of symmetric properties with the usual meaning; if a property p holds for a pair (x, y), then it also holds for (y, x).

Inverse properties. (`owl:inverseOf`) Furthermore, properties can be stated to be the inverse of another property, i.e. hasParent and hasChild. If p_1 is the inverse of p_2 and an individual x is related to another individual y by p_1, then y is related by p_2 to x.

Property equivalence. (`owl:equivalentProperty`) Two properties may be stated to be equivalent in the same fashion as classes can. Equivalent properties relate one individual to the identical set of other individuals.

Individual equivalence. (`owl:sameAs`) Individual equality is included in the language for practical purposes since two distinct URIs can identify the same resource. While individual equality slightly raises the computational complexity (see [14] for an in-depth treatment) it can still be dealt with in practical implementations by various means.

4 Formal Semantics

4.1 Basic Definitions

In this section we give a formal definition of the language primitives outlined in the previous section using specific *entailment rules*. To do so, we briefly recall the required basic terminology as in [15], as a slight extension of the notions in [16].

First, let U denote the set of *URI references*, B denote the (infinite) set of *blank nodes*, and L denote the set of literals, i.e. data values such as strings, booleans, or XML documents. L is partitioned into the set L_p of *plain literals* and the set L_t of *typed literals*. A *typed literal* l consists of a lexical form s and a datatype URI t; l can then be denoted as the pair $l = (s, t)$. The sets U, B, L_p, and L_t are pairwise disjoint. A *vocabulary* is a subset of $U \cup L$. Any symbol t in $U \cup B \cup L$ is called a *RDF term* and the set of RDF terms is denoted by T.

The basic notion of RDF graphs [17,16] only allows URI references in the place of predicates, however, *generalized RDF graphs*, which also allow properties to be blank nodes, were introduced in [15] to solve the problem that the standard set of entailment rules for RDFS [17] is incomplete.

Definition 1 (Generalized RDF Graph). *An RDF graph G is a subset of the set* $(U \cup B) \times (U \cup B) \times (U \cup B \cup L)$.

The elements (s, p, o) of an RDF graph are called *triples*, which consist of a subject s, a predicate (or property) p, and an object o, respectively. We write triples as `s p o .`

The set $T(G)$ of *RDF terms of an RDF graph G* is the set of all elements that occur in the graph, and the set $bl(G)$ of *blank nodes of an RDF graph G* is in turn defined as $bl(G) = T(G) \cap B$. A graph is ground if it does not contain any blank nodes, that is if $bl(G) = \emptyset$.

Definition 2 (Vocabulary of an RDF graph). *Based on this, the* vocabulary of an RDF graph G *is defined by* $V(G) = T(G) \cap (U \times L)$.

An interpretation of an RDF graph is intrinsically tied to this notion of a specifc *vocabulary* (RDF, RDFS, ...), as in [17], starting with *simple interpretation*, as following:

Definition 3 (Simple Interpretation). *An interpretation* I *of a vocabulary* V *is a tuple* $I = (R_I, P_I, E_I, S_I, L_I, LV_I)$*, where* R_I *is a nonempty set, called the set of* resources*,* P_I *is the set of* properties *(not required to be disjoint from resources),* LV_I *is the set of* literal values*, which is a subset of* R_I *that contains at least all plain literals in* V*, and where* E_I *,* S_I *and* L_I *are functions:*

- $E_I : P_I \rightarrow 2^{R_I \times R_I}$
- $S_I : (V \cap U) \rightarrow (R_I \cup P_I)$
- $L_I : (V \cap L_t) \rightarrow R_I$

4.2 Entailment Rules

We then use *entailment rules*, as in [15]. An entailment rule is considered as a pair of generalized RDF graphs where variables can occur as predicate, subject and object in triples. In other words, a rule consists of two sets of triple patterns[6].

For any rule $\rho = (\rho_l, \rho_r)$, we call ρ_l the body of the rule ρ and ρ_l the head of the rule. Syntactically such rules take the following simple form:

$$\texttt{IF } \rho_l \texttt{ THEN } \rho_r$$

Informally, a proper entailment rule describes under which conditions ρ_l the statements ρ_r must hold. From this, the statements ρ_r can be inferred whenever we detect the situation specified by ρ_l – it characterizes the expected inferences over a domain vocabulary.

Given a rule ρ, the set of *variables of* ρ is denoted by $var(\rho) = var(\rho_l)$, the set of *blank nodes of* ρ by $bl(\rho) = bl(\rho_r)$, and the *vocabulary of* ρ by $V(\rho) = V(\rho) \cup V(\rho_r)$.

If R is a set of rules, then $V(R) = \bigcup_{\rho \in R} V(\rho)$. An entailment rule ρ is said to *introduce blank nodes* if $bl(\rho) \neq \emptyset$. A rule ρ is called *finite* if the rule head ρ_r and the rule body ρ_l are both finite. A rule ρ is called a *proper rule* if the rule head ρ_r and the rule body ρ_l are both nonempty.

From the above, it is possible to define the meaning of entailment rules in a model-theoretic sense, by defining when a rule is satisfied by an interpretation, and secondly by defining what statements (triples) are entailed by a specific set of rules R, i.e. the notion of simple R-entailment.

[6] In the sense defined by the RDF Data Access Group, W3C, `http://www.w3.org/2001/sw/DataAccess/`

4.3 Definition of *L2* Language Features

We are now in a position to give a concise, formal definition of the semantics of *L2* by defining (i) its vocabulary, and (ii) the corresponding set of entailment rules, as described in the previous sections. *L2*'s vocabulary is constructed as an extension of the RDF and RDFS vocabulary (see [17]) and adds the following selected constructs from OWL:

Definition 4 (*L2* Vocabulary). $V_{L2} = \{$ `owl:sameAs`, `owl:SymetricProperty`, `owl:TransitiveProperty`, `owl:inverseOf`, `owl:equivalentClass`, `owl:equivalentProperty` $\} \cup V_{RDFS} \cup V_{RDF}$

L2's set of entailment rules is then defined on top of RDFS entailment (omitting literal generalization) and several additional rules covering the OWL primitives as depicted in Table 1. The semantics defined for them via the listed entailment rules are slightly weaker than their OWL counterparts, mostly for performance reasons, and in this sense *L2* is a semantic subset. In the following, we point out some important characteristics of the chosen rule set.

- For performance reasons *L2* has only "if-conditions" for e.g. `rdf:range`, `rdf:domain`, `rdf:subClassOf`, `rdf:subPropertyOf`, `owl:TransitiveProperty`, etc. instead of the stronger extensional "if and only if conditions" as in OWL.
- In order to capture the intended semantics of class and property hierarchies, including reflexivity and transitivity, rules are included to make this notion explicitly visible.
- Axiomatic triples are not considered during inference.
- Class equivalence is cleanly layered on top of RDFS in the sense that two classes are considered equivalent if and only if they are both a subclass of each other, whereas in OWL only their extensions have to be equal. The same reasoning applies for property equivalence. This style of modeling the semantics of equivalence is rooted in the fact that equivalence, e.g. between classes, can already be indirectly expressed in RDFS in this way, only the vocabulary to make this explicit was not available.
- Furthermore OWL treats `owl:sameAs` strictly as equivalence whereas *L2* slightly weakens its interpretation and only treats it as an equivalence relation. In order to recapture a set of essential inferences several additional rules are added.

Common reasoning tasks, such as query answering, reduce to entailment between two generalized RDF graphs. Due to its close relationship with pD* [10] known complexity and tractability carry over to *L2*, i.e. ground entailment can be checked in polynomial time. Moreover, we ensure tractability by restricting entailment rules to Horn rules (see [18] for relevant complexity results).

For the specific rule-set of *L2* we additionally give relevant complexity measures in Table 1. These include for each rule, the time complexity $\mathcal{T}$ for detecting a required rule application and the space complexity Δ for the number of triples inferred (the number of nodes needed to construct the closure graph in terms

Table 1. Intended semantics for $L2$ given by means of first-order implications / entailment rules. Rule (1) and (2) cover symmetry and transitivity of properties. Rules (3a) and (3b) formalize the notion that an individual can be considered to be equal to itself. Rule (4) captures reflexivity and respectively and rule (5) transitivity of individual equivalence. Rule (6) and (7) cover the semantics of inverse properties, including its reflexivity. Rules (8) and (9) denote that individuals that are classes or properties are considered sub-classes or sub-properties of themselves. These rules are important to facilitate basic meta-modelling in the language. Rule (10) denotes that existing relations are preserved when renaming nodes. Rules (11a), (11b) and (11c) express the semantics of class equivalence, while (12a), (12b) and (12c) do the same for property equivalence.

Rule No.	IF	THEN	$\mathcal{T}$	Δ
1	?p `type SymmetricProperty` ?v ?p ?w	?w ?p ?v	$O(n^2)$	$O(n)$
2	?p `type TransitiveProperty` ?u ?p ?v ?v ?p ?w	?u ?p ?w	$O(n^3)$	$O(n^2)$
3a	?v ?p ?w	?v `sameAs` ?v	$O(n)$	$O(n)$
3b	?v ?p ?w	?w `sameAs` ?w	$O(n)$	$O(n)$
4	?v `sameAs` ?w	?w `sameAs` ?v	$O(n)$	$O(n)$
5	?u `sameAs` ?v ?v `sameAs` ?w	?u `sameAs` ?w	$O(n^2)$	$O(n^2)$
6	?p `inverseOf` ?q ?v ?p ?w	?w ?q ?v	$O(n^2)$	$O(n)$
7	?p `inverseOf` ?q ?v ?q ?w	?w ?p ?v	$O(n^2)$	$O(n)$
8	?v `type Class` ?v `sameAs` ?w	?v `subClassOf` ?w	$O(n^2)$	$O(n)$
9	?p `type Property` ?p `sameAs` ?q	?p `subPropertyOf` ?q	$O(n^2)$	$O(n)$
10	?u ?p ?v ?u `sameAs` ?w ?v `sameAs` ?q	?w ?p ?q	$O(n^3)$	$O(n)$
11a	?v `equivalentClass` ?w	?v `subClassOf` ?w	$O(n)$	$O(n)$
11b	?v `equivalentClass` ?w	?w `subClassOf` ?v	$O(n)$	$O(n)$
11c	?v `subClassOf` ?w ?w `subClassOf` ?v	?v `equivalentClass` ?w	$O(n^2)$	$O(n)$
12a	?v `equivalentProperty` ?w	?v `subProperty` ?w	$O(n)$	$O(n)$
12b	?v `equivalentProperty` ?w	?w `subProperty` ?v	$O(n)$	$O(n)$
12c	?v `subPropertyOf` ?w ?w `subPropertyOf` ?v	?v `equivalentProperty` ?w	$O(n^2)$	$O(n)$

Table 2. Omitted rules and the associated scalability with respect to the to the increase in the size of the computed closure and the effort needed to apply them

Rule No.	IF	THEN	$\mathcal{T}$	Δ
N1	?p `type FunctionalProperty` ?u ?p ?v ?u ?p ?w	?v `sameAs` ?w	$O(n^3)$	$O(n)$
N2	?p `type InverseFunctionalProperty` ?u ?p ?w ?v ?p ?w	?u `sameAs` ?w	$O(n^3)$	$O(n)$
N3	?v `hasValue` ?w ?v `onProperty` ?p ?u ?p ?w	?u `type` ?w	$O(n^3)$	$O(n)$
N4	?v `hasValue` ?w ?v `onProperty` ?p ?u `type` ?v	?u ?p ?w	$O(n^3)$	$O(n)$
N5	?v `someValuesFrom` ?w ?v `onProperty` ?p ?u ?p ?x ?x `type` ?w	?u `type` ?v	$O(n^4)$	$O(n)$
N6	?v `allValuesFrom` ?w ?v `onProperty` ?p ?u `type` ?v ?u ?p ?x	?x `type` ?w	$O(n^4)$	$O(n)$

of the size of the initial graph). To contrast this with more computationally expensive entailment rules, we show the same information for additional rules from [10] in Table 2.

As shown the highest time complexity for the rules we included in $L2$ is $O(n^3)$, whereas it is $O(n^4)$ for the omitted rules in Table 2. The most complex rule covers transitive properties (Rule 2), which poses the same challenges as existing RDFS vocabulary. As a practical solution, the application of this rule on a graph can be mapped to a well studied problem, graph reachability, where efficient optimization algorithms exist see [19] [20] [21].

5 Conclusion

In this paper we presented $L2$, a lightweight and tractable language for the description of resources on the Semantic Web for which rule based and efficient reasoning methods are directly applicable. For that purpose we considered related work concerning theoretical research results as well as practical implementations that are similar in spirit to our approach. We gave a high level explanation of the modeling primitives supported, that (i) are implementable in a scalable way and (ii) useful in practical settings. Lastly, we gave a formal definition of entailment rules that capture the semantics of $L2$ and from which it is straightforward to establish the tractability of $L2$.

It should be noted, that the definition of the formal semantics of *L2* by restricted entailment rules is not the only possible approach and should not necessarily be taken as a direct algorithmic evaluation procedure. However, this approach can be understood as a basis for defining a minimal, useful and implementable language that is in line with existing Web standards, and also allows for extension with custom rule sets.

Acknowledgments

This research has been partially supported by the LarKC EU-funded project (FP7-215535). For more information visit `http://www.larkc.eu`.

References

1. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The Semantic Web. Scientific American 284(5), 28–37 (2001)
2. Bizer, C., Heath, T., Ayers, D., Raimond, Y.: Interlinking Open Data on the Web. In: Demonstrations Track, 4th European Semantic Web Conference, Innsbruck, Austria (2007)
3. McGuinness, D., van Harmelen, F., et al.: OWL Web Ontology Language Overview. W3C Recommendation 10, 2004–03 (2004)
4. Brachman, R., Levesque, H.: The tractability of subsumption in frame-based description languages. In: Proc. of the 4th Nat. Conf. on Artificial Intelligence (AAAI 1984), pp. 34–37 (1984)
5. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation 2 (2004)
6. Wang, T., Parsia, B., Hendler, J.: A Survey of the Web Ontology Landscape. In: Cruz, I., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L.M. (eds.) ISWC 2006. LNCS, vol. 4273, pp. 682–694. Springer, Heidelberg (2006)
7. Weithoner, T., Liebig, T., Luther, M., Bohm, S.: What's Wrong with OWL Benchmarks? In: Second International Workshop on Scalable Semantic Web Knowledge Base Systems, SSWS 2006 (2006)
8. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: DL-Lite: Tractable Description Logics for Ontologies. In: Proceedings of the National Conference on Artificial Intelligence, vol. 20(2), p. 602 (2005)
9. Baader, F., Brandt, S., Lutz, C.: Pushing the EL Envelope Further. In: Proceedings of the OWLED Workshop (2008)
10. ter Horst, H.J.: Combining RDF and part of owl with rules: Semantics, decidability, complexity. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 668–684. Springer, Heidelberg (2005)
11. Krötzsch, M., Rudolph, S., Hitzler, P.: Elp: Tractable rules for owl 2. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 649–664. Springer, Heidelberg (2008)
12. Grau, B., Horrocks, I., Parsia, B., Patel-Schneider, P., Sattler, U.: Next Steps for OWL. OWL Experienced and Directions (2006)
13. Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang, H.: The Manchester OWL Syntax

14. Volz, R.: Web Ontology Reasoning with Logic Databases. PhD thesis, Universität Karlsruhe (TH), Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe (2004)
15. ter Horst, H.J.: Completeness, decidability and complexity of entailment for RDF schema and a semantic extension involving the owl vocabulary. J. Web Sem. 3(2-3), 79–115 (2005)
16. Klyne, G., Carroll, J., McBride, B.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 (2004)
17. Hayes, P., McBride, B.: RDF Semantics. W3C Recommendation 10 (2004)
18. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys (CSUR) 33(3), 374–425 (2001)
19. Schenkel, R., Theobald, A., Weikum, G.: Efficient Creation and Incremental Maintenance of the HOPI Index for Complex XML Document Collections. In: Proceedings of the International Conference on Data Engineering, 1998, vol. 21, p. 360. IEEE Computer Society Press, Los Alamitos (2005)
20. Schenkel, R., Theobald, A., Weikum, G.: HOPI: An Efficient Connection Index for Complex XML Document Collections. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K., Ferrari, E. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 237–255. Springer, Heidelberg (2004)
21. Wang, H., He, H., Yang, J., Yu, P., Yu, J.: Dual labeling: Answering graph reachability queries in constant time. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE), p. 75 (2006)