Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary ⋆

Herman J. ter Horst

Philips Research, Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands

Abstract

We prove that entailment for RDFS (RDF Schema) is decidable, NP-complete, and in P if the target graph does not contain blank nodes. We show that the standard set of entailment rules for RDFS is incomplete and that this can be corrected by allowing blank nodes in predicate position.

We define semantic extensions of RDFS that involve datatypes and a subset of the OWL vocabulary that includes the property-related vocabulary (e.g. `FunctionalProperty`), the comparisons (e.g. `sameAs` and `differentFrom`) and the value restrictions (e.g. `allValuesFrom`). These semantic extensions are in line with the 'if-semantics' of RDFS and weaker than the 'iff-semantics' of D-entailment and OWL (DL or Full). For these semantic extensions we present entailment rules, prove completeness results, prove that consistency is in P and that, just as for RDFS, entailment is NP-complete, and in P if the target graph does not contain blank nodes. There are no restrictions on use to obtain decidability: classes can be used as instances.

Key words: ontology, semantics, entailment, completeness, computational complexity

1 Introduction and Overview

One of the main purposes of the Semantic Web [2] is to 'make reasoning explicit': statements that enable conclusions to be drawn are to be represented

⋆ This paper is a revised and extended version of the paper entitled: "Extending the RDFS Entailment Lemma", which was presented at the Third International Semantic Web Conference (ISWC2004) [10].

Email address: `herman.ter.horst@philips.com` (Herman J. ter Horst).

explicitly on the web, for machine use. The W3C has defined two languages for the Semantic Web: RDF and OWL. RDF (Resource Description Framework) [15] plays a basic role by allowing the expression of statements, in the form of subject-predicate-object triples. OWL (Web Ontology Language) [13] allows the expression of ontologies, which define the meaning of terms used in RDF statements. Simple ontologies can already be expressed using the RDF Schema (RDFS) vocabulary [15]. For RDF, RDFS and OWL, the valid conclusions, called entailments, are determined by the semantics of the language [8,14]. The RDFS entailment lemma [8] claims completeness of a set of simple entailment rules for RDFS. This result is used to implement reasoners and is useful for clarifying to people, in syntactic terms, what RDFS entailment is. The RDFS entailment lemma does not show decidability of RDFS entailment because the closure graphs used in its proof are infinite for finite RDF graphs. For OWL, no complete set of entailment rules is known.

This paper has several purposes. It aims to:

- settle the fundamental question of decidability of RDFS entailment,
- determine the computational complexity of RDFS entailment, and
- extend the semantics and the complete set of simple entailment rules for RDFS, with the results relating to decidability and complexity, so as to apply to the OWL vocabulary.

In order to achieve the first and second aims, we prove a refined completeness result for RDFS, which uses partial closure graphs that can be taken to be finite for finite RDF graphs and can be computed in polynomial time. To achieve the third aim, we also consider the inclusion of datatypes and describe a non-standard semantics that applies to a subset of the OWL vocabulary. This non-standard semantics, called the pD* semantics, is defined in a way analogous to the semantics of RDFS, but is weaker than the standard semantics of OWL. pD* entailment shares with RDFS entailment a relatively low computational complexity (NP, and in a non-trivial case P) and the availability of a complete set of simple entailment rules. This leads to a relatively low threshold for implementation.

This paper also aims to give a mathematically clear account of the results, including complete proofs. Compared with [8], this paper devotes more attention to the mathematical aspects of the RDF semantics. While writing this paper, I discovered that the standard set of entailment rules for RDFS is incomplete and that this can be corrected by allowing blank nodes in predicate position.

This paper is organized as follows. The remainder of Section 1 provides an overview and discussion of the results of this paper and some background, without going into mathematical details. In particular, we discuss in 1.8 why

it is reasonable, apart from complexity considerations, to consider a weakened semantics in connection with the OWL vocabulary. Section 2 summarizes terminology and notation relating to RDF graphs, defines generalized RDF graphs which allow blank nodes in predicate position, extends the interpolation lemma to this setting, and proves that simple entailment is NP-complete. Section 3 defines the D* semantics, which generalizes the RDFS semantics to include reasoning with datatypes. Section 4 proves that RDFS entailment is decidable, NP-complete, and in P if the target graph does not contain blank nodes. At the same time, these results are proved for an extension with datatypes, by means of the D* entailment lemma. This is a completeness result which generalizes the RDFS entailment lemma. Section 5 describes the pD* semantics and extends the completeness, decidability and complexity results for RDFS to pD* entailment. Section 6 extends the completeness result for pD* entailment to incorporate OWL's complete iff condition for `someValuesFrom`. Section 7 summarizes the conclusions. In comparison with the conference version [10], this paper considers an enlarged subset of the OWL vocabulary, removes an assumption from the results (the assumption that datatype maps are discriminating) and completes the proofs of the results. Moreover, this paper adds to [10] several NP-completeness proofs (announced in the presentation of [10] and recorded in Propositions 2.13, 4.14 and 5.15, which originally appeared in [11]) and the overview that follows below.

1.1 RDF and OWL: semantics. Although the standard syntax for RDF and OWL uses XML, it should be noted that the meaning of RDF and OWL knowledge bases is independent of XML and abstracts from the XML serialization used. Here the notion of RDF graph [12] plays a role. An RDF graph is a set of RDF statements, i.e. subject-predicate-object triples; subjects and objects of triples are viewed as nodes, linked by predicates (predicates are usually called properties). For RDF, RDFS (RDF Schema) and OWL, the entailments (i.e. the valid conclusions) are determined by the semantics, which is specified by means of a model theory [8,14]. These model-theoretic specifications follow a general pattern that is derived from mathematical logic [3]. For an entailment relation X (e.g. RDFS entailment), an RDF graph S is said to X-entail an RDF graph G if each X-interpretation that satisfies S also satisfies G. This definition is made complete with a detailed, mathematical definition of X-interpretation. For RDF and OWL, interpretations can be infinite and there are infinitely many interpretations. Therefore, to make the connection with reasoners and computation, a bridge needs to be constructed. Before summarizing what is known about the computational aspects of the entailment relations specified for RDF and OWL, we mention a fundamental distinction between two different ways of specifying semantic conditions.

1.2 Semantics: 'if' conditions versus 'iff' conditions. RDF and OWL differ in the way in which their semantics is defined [8,14]. The semantics of RDF and RDFS is defined using if conditions, whereas the semantics of OWL

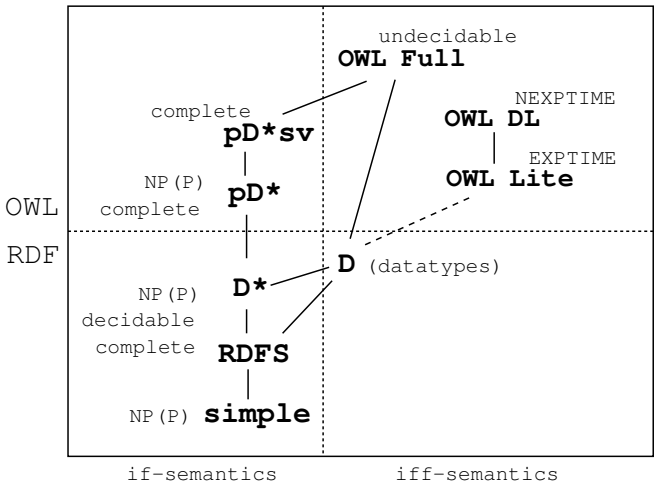

Fig. 1. Entailment relations for RDF and OWL: overview. Entailment relations labeled 'complete' have a complete set of entailment rules.

uses many if-and-only-if conditions. [1] A semantics that uses iff conditions in its specification is more powerful, in the sense that it leads to more entailments. The distinction can be illustrated using the notion `subClassOf` from the RDFS (RDF Schema) vocabulary. If c and d are classes, then RDFS requires that if c is a subclass of d, then the extension of c (i.e. the set of instances of c) is a subset of the extension of d. On the other hand, OWL DL and OWL Full require that c is a subclass of d if and only if the extension of c is a subset of the extension of d. This example already indicates that a semantics that uses iff conditions in its specification is indeed stronger: an iff-semantics may lead to entailments that are not supported under an if-semantics, stating that a class is a subclass of another class.

1.3 RDF and OWL: overview of entailment relations. Figure 1 presents an overview of entailment relations for RDF and OWL. The RDFS semantics is specified by means of if conditions [8], as was already mentioned. RDFS entailment extends another entailment relation, called simple entailment, which deals specifically with blank nodes, i.e. variables. The specifications of the three standard versions of the OWL semantics, OWL Full, OWL DL and OWL Lite, include many iff conditions [14]. The RDF specification contains another entailment relation, for reasoning with datatypes, called D-entailment [8]. D-entailment extends RDFS entailment, while the OWL entailment relations in turn extend D-entailment. Although D-entailment has been placed on the iff side in Figure 1, it is actually specified by means of if conditions in combination with just one iff condition; the latter condition requires that an element of the universe of interpretation is a value of a datatype if and only if it is an instance of this datatype regarded as a class.

[1] In connection with the semantics of RDFS, if conditions have been called intensional conditions, while iff conditions have been called extensional conditions [8].

The meaning of the lines in Fig. 1 can be described as follows. If an entailment relation X is connected with a line to an entailment relation Y lower in the figure, then X extends Y (i.e., each Y-entailment is also an X-entailment) and is stronger than Y (i.e., there is an X-entailment that is not an Y-entailment). However, it should be noted that although there is an RDF-compatible semantics defining OWL DL (and OWL Lite) as an extension of D-entailment (see [14], Section 5), OWL DL is normatively defined by means of a direct model-theoretic semantics (see [14], Section 3), independently of D-entailment. The line between D-entailment and OWL Lite in Figure 1 has been drawn differently from the other lines in view of the fact that OWL DL and OWL Lite impose certain restrictive assumptions not used for D-entailment and OWL Full. For example, OWL DL and OWL Lite do not allow the use of classes as instances.

What is known about these standard entailment relations from the RDF and OWL specifications? For RDFS entailment there is a standard set of simple entailment rules, which is claimed to be complete by the RDFS entailment lemma [8]. OWL Full entailment is known to be undecidable [9]. For OWL DL, the restrictive assumptions already mentioned have been placed on the use of the language to ensure decidability. With regard to computational complexity, OWL DL entailment is known to be complete for NEXPTIME, while entailment for the OWL Lite part is known to be complete for EXPTIME [9]. These are worst-case complexity results. Optimized reasoners exist; these make use of techniques developed for description logics [1]. For D-entailment no complete set of entailment rules is known and decidability is unknown.

The main results of this paper can be summarized as follows. RDFS entailment is decidable, NP-complete, and in P if the target graph does not contain blank nodes. There is a somewhat weaker variant of D-entailment that extends RDFS entailment with reasoning with datatypes, called D* entailment, that is defined by means of if conditions and has the same computational properties as RDFS entailment: there is a complete set of simple entailment rules, it is decidable, NP-complete or even in P. A new entailment relation is considered in the upper left quadrant of Figure 1: there is a variant of OWL entailment specified by means of if conditions, again with the same computational properties as RDFS entailment. The notion of pD* entailment extends D* entailment and includes semantic conditions for a subset of the OWL vocabulary. We also show that for the RDFS, D* and pD* semantics, consistency is in P. We extend the completeness result for the pD* semantics to the pD*sv semantics, which incorporates OWL's complete iff condition for `somevaluesFrom`. Finally, we show that the standard set of entailment rules for RDFS is incomplete and that this can be corrected by allowing blank nodes in predicate position. We consider the left part of Figure 1 more closely, starting at the bottom and proceeding upward.

1.4 Simple entailment. RDFS entailment extends simple entailment, which handles blank nodes. Blank nodes are variables, which can be used as subjects and objects of RDF statements. Blank nodes are, implicitly, existentially quantified. For example, the RDF statement

```
Iliad hasAuthor Homer .
```

simply entails the RDF statement

```
Iliad hasAuthor b .
```

Here b is assumed to be a blank node (in this example and several other examples, we use the N-Triples syntax for RDF [5], in abbreviated form, using meta-variables such as b in the example just given). The intuition behind simple entailment is that replacements can be found for the blank nodes in the target graph in a way that is consistent with the given graph. In this paper we prove that the problem S simply-entails G, with S and G finite RDF graphs, is NP-complete. If the target graph G does not have blank nodes, this problem is in P, as it is only necessary to check that each triple in G is also in S.

1.5 RDFS entailment: completeness. In [8] the RDFS entailment lemma is presented, which aims to provide a complete reasoning procedure for RDFS entailment, reflecting in syntactic terms exactly what RDFS entailment is. According to the RDFS entailment lemma, any RDFS entailment can be obtained with a finite derivation by means of simple entailment rules and axiomatic triples. There are 18 entailment rules and infinitely many axiomatic triples. As an example, for RDF's standard container types (e.g. sequence and bag) the container membership properties `rdf:`$\_i$ are used as well as axiomatic triples such as

`rdf:`$\_i$ `rdf:type rdf:Property .`

This triple specifies `rdf:`$\_i$ to belong to the standard class `rdf:Property`. Here i can have any value 1,2,..., which shows that there are infinitely many axiomatic triples. We give three examples of rules. According to entailment rule rdfs9, any two RDF triples of the form

u `rdf:type` v .
v `rdfs:subClassOf` w .

(i.e. u is of type v and v is a subclass of w) entail the triple

u `rdf:type` w .

According to entailment rule rdfs2, any two RDF triples of the form

v p w .
p `rdfs:domain` u .

entail the triple

v `rdf:type` u .

According to entailment rule rdfs7, two RDF triples of the form

v p w .
p `rdfs:subPropertyOf` q .

entail the triple

v q w .

It turns out that the completeness claim of the RDFS entailment lemma is not correct, as was overlooked in [8] and [10]. We describe an example, involving the entailment rules rdfs2 and rdfs7 just mentioned. Consider the RDF graph consisting of the following three triples:

p `rdfs:subPropertyOf` b . (1)
b `rdfs:domain` u . (2)
v p w . (3)

Here b is assumed to be a blank node. It is not difficult to prove by means of the semantic definitions that these three triples RDFS-entail the triple

v `rdf:type` u . (4)

(See 4.3 for the proof.) Although the semantics of RDFS has a 'higher-order' aspect, allowing blank nodes to refer to classes and properties, the syntax does not allow blank nodes to appear in predicate position. In view of this syntactic restriction, the triple (4) cannot be derived from the triples (1), (2), (3) by means of the entailment rules for RDFS. Rule rdfs7, for `rdfs:subPropertyOf`, cannot be applied to the triples (1) and (3) to obtain the triple

v b w . (5)

which would be needed to complete the proof of (4) by using triple (2) and rule rdfs2 for `rdfs:domain`.

In this paper we extend the notion of RDF graph to allow blank nodes in predicate position and show that the standard entailment rules for RDFS become complete if rule rdfs7 is replaced by a suitably extended rule rdfs7x. In view of the fact that the semantics of RDFS allows blank nodes to refer to properties, 'generalized RDF graphs', which allow blank nodes in predicate position, seem to form a more direct abstract syntax for RDFS than RDF graphs.

If b is a blank node that does not appear in other triples of an RDF graph, then the triple

$$v \;\; b \;\; w \;\; . \tag{6}$$

expresses that there exists a relation between v and w, although nothing is expressed about this relation. It is interesting to note that in this generalized setting for RDF, the question of whether items v and w are related when an RDF graph is given can be answered by checking whether the RDF graph entails the triple (6).

1.6 RDFS entailment: decidability, complexity. The RDFS entailment lemma does not show decidability of RDFS entailment. The existence of a complete reasoning procedure in itself does not guarantee decidability; compare first-order logic, with Gödel's completeness theorem and Church's theorem showing undecidability [3]. Moreover, the proof of the RDFS entailment lemma [8] makes use of closure graphs which are infinite for finite graphs. The RDFS closure of an RDF graph is formed by first adding all (infinitely many) axiomatic triples to the graph and by subsequently applying the entailment rules until the graph is unchanged. An RDFS closure may contain an XML clash, which is an easily identifiable kind of triple resulting from ill-formed XML used as object value in an RDF statement (this will be discussed further in 1.7). According to the RDFS entailment lemma, an RDF graph S RDFS-entails an RDF graph G if and only if the RDFS closure of S simply entails G (or contains an XML clash). This implies that in order to decide whether S RDFS-entails G, it is sufficient to check whether a finite subgraph of the RDFS closure of S simply entails G. However, since the closure of S is infinite, and therefore the number of candidate subgraphs is infinite, this is not a finite algorithm.

We prove decidability of RDFS entailment by refining the RDFS entailment lemma. Instead of using a full RDFS closure, we show that it is sufficient to use a partial RDFS closure which excludes the axiomatic triples for the container membership properties `rdf:`$\_i$ not used in either the source graph S or the target graph G, as long as these axiomatic triples are included for at least one property `rdf:`$\_i$. According to the version of the RDFS entailment lemma proved in this paper, an RDF graph S RDFS-entails an RDF graph G

if and only if such a partial closure of S simply entails G (or contains an XML clash). This refined version of the RDFS entailment lemma leads to most of the technical complexity of the proofs in this paper. It implies decidability of RDFS entailment. We show that the partial closures used can be computed in polynomial time and that, just as for simple entailment, the problem 'S RDFS-entails G' is in NP, and in P if G does not have blank nodes; we also prove that this problem (without restrictive assumptions) is NP-complete. The partial closures that we use are, in general, 'generalized RDF graphs', which allow blank nodes in predicate position (cf. 1.5).

The result that the problem 'S RDFS-entails G' is in P if G does not have blank nodes is relevant in practice. The target graph G typically forms a relatively small part of the combined problem data S, G, and may contain relatively few blank nodes. The results clearly indicate the computational cost of using blank nodes in a target graph. [2]

1.7 Inclusion of datatypes: D* entailment. RDFS entailment includes conditions for a special datatype, `rdf:XMLLiteral`, which enables pieces of XML text to be used as data values in RDF statements. By applying these semantic conditions for the datatype `XMLLiteral` to other datatypes as well, a more general entailment relation is obtained, which is called D* entailment and which extends RDFS entailment to include reasoning with datatypes. It should be noted that the treatment of `rdf:XMLLiteral` in RDFS is different from the treatment of datatypes in datatyped interpretations, and that D* entailment is weaker than the standard notion of D-entailment [8] which extends RDFS with datatyped reasoning. With respect to D* entailment (and D-entailment), the datatypes used are combined in a datatype map, which can include, for example, the standard XML Schema datatypes `xsd:string` and `xsd:integer` required for OWL entailment. D* entailment has the same computational properties as RDFS entailment: there is a complete set of simple entailment rules, it is NP-complete, and in P if the target graph has no blank nodes. These results are obtained with only one change to the entailment rules for RDFS. [3] To explain the other changes required to go from RDFS entailment to D* entailment, we mention two further entailment rules for RDFS. Rule lg (literal generalization) states that an RDF triple of the form

$$v \;\; p \;\; l \;\; . \tag{7}$$

[2] There seems to be little theoretical work on the RDF model and its semantic extensions. An exception is [6], which focuses on fundamental database questions (redundancy, queries). In [6] it is also stated, without proof, that a simplified variant of RDFS entailment, which ignores the semantic conditions leading to infinite closures, is NP-complete.

[3] See Table 4 for the complete set of 18 entailment rules for RDFS entailment and D* entailment, including the conditions of applicability.

where l is a literal (i.e. a data value), entails the triple

v p b_l .

where b_l is a new, surrogate blank node for l. These surrogate blank nodes are used, for example, in rule rdf2, which states that the same given triple (7), where l is now a well-typed XML literal, entails the triple

b_l `rdf:type rdf:XMLLiteral` .

As was already mentioned in 1.6, an RDFS closure or a partial RDFS closure may contain an XML clash: an XML clash is a triple

b_l `rdf:type rdf:Literal` .

where b_l is the blank node allocated by rule lg to an ill-typed XML literal. To go from RDFS entailment to D* entailment, the entailment rule rdf2 just mentioned is replaced by rule rdf2-D, with the same effect as rule rdf2 not only for `XMLLiteral` but for all datatypes in a datatype map. In addition, the notion of XML clash is replaced by the analogous notion of D-clash which applies not only to ill-typed XML literals but to ill-typed literals of any type in the given datatype map. While XML clashes are the only inconsistencies that can arise under the RDFS semantics, D-clashes are the only inconsistencies that can arise under the D* semantics. For example, if the given datatype map contains the standard XML Schema datatypes string and integer, then the two triples

b `rdf:type xsd:string` . (8)
b `rdf:type xsd:int` . (9)

where b is a blank node, are consistent under the D* semantics. On the other hand, these two triples are inconsistent under the D-semantics [8], since the value spaces of the two datatypes string and integer are disjoint. This shows that D-entailment is indeed stronger than D* entailment. Consider, however, the following meta-modeling triple, which uses the OWL vocabulary:

`xsd:string owl:disjointWith xsd:int` . (10)

Under the pD* semantics, to which we turn next, the triples (8) and (9), together with the triple (10), are inconsistent.

1.8 pD* entailment. By weakening the standard iff-semantics of OWL, we define the pD* entailment relation, which extends RDFS entailment

and D* entailment, is largely defined by means of if conditions, and applies to a property-related subset of the OWL vocabulary. This subset includes `FunctionalProperty`, which has been stated by several people to be the single most useful element of the OWL vocabulary. In addition to `FunctionalProperty`, the pD* semantics has semantic conditions for `InverseFunctionalProperty`, `sameAs`, `SymmetricProperty`, `TransitiveProperty`, `inverseOf`, `equivalentClass`, `equivalentProperty`, `hasValue`, `someValuesFrom`, `allValuesFrom`, `differentFrom` and `disjointWith`. With respect to this subset of the OWL vocabulary, the pD* semantics is intended to represent a reasonable interpretation that is useful for drawing conclusions about instances in the presence of an ontology, and that leads to simple entailment rules and a relatively low computational complexity. We show completeness of a set of simple entailment rules for pD* entailment. [4] Just like RDFS entailment and D* entailment, pD* entailment is decidable, NP-complete, and in P if the target graph has no blank nodes. Just as for RDFS entailment and D* entailment, a partial closure that is sufficient for deciding entailment can be computed in polynomial time.

As an example of an entailment rule, rule rdfp1 states that three triples of the form

p `rdf:type owl:FunctionalProperty` .
u p v .
u p w .

entail the triple

v `owl:sameAs` w .

As another example, rule rdfp4 states that three triples of the form

p `rdf:type owl:TransitiveProperty` .
u p v .
v p w .

entail the triple

u p w .

[4] The complete set of 18 entailment rules for RDFS entailment and D* entailment shown in Table 4 can be combined with 23 other rules to form a complete set for pD* entailment. See Table 7 for these rules.

For `hasValue`, the complete if-and-only-if condition of OWL's semantics is included in the pD* semantics. For `allValuesFrom` and `someValuesFrom`, an if-semantics is provided, realizing half of OWL's iff conditions. The entailment rule for `allValuesFrom` (rdfp16) essentially states that, for an instance of an `allValuesFrom` restriction with respect to a certain class and a certain property, each value of the property can be concluded to belong to the class. More explicitly, rule rdfp16 states that four triples of the form

$$v \texttt{ owl:allValuesFrom } w \texttt{ .} \tag{11}$$
$$v \texttt{ owl:onProperty } p \texttt{ .} \tag{12}$$
$$u \texttt{ rdf:type } v \texttt{ .}$$
$$u \; p \; x\texttt{.}$$

entail the triple

$$x \texttt{ rdf:type } w \texttt{ .}$$

Here the two triples (11) and (12) define the `allValuesFrom` restriction class v with respect to the class w and the property p.

The entailment rule for `someValuesFrom` (rdfp15) is formulated the other way around: it essentially states, for a certain given entity, that if a value of a certain property belongs to a certain class, then this entity can be concluded to belong to the `someValuesFrom` restriction with respect to this property and class when this `someValuesFrom` restriction has been defined. More explicitly, rule rdfp15 states that four triples of the form

$$v \texttt{ owl:someValuesFrom } w \texttt{ .}$$
$$v \texttt{ owl:onProperty } p \texttt{ .}$$
$$u \; p \; x \texttt{ .}$$
$$x \texttt{ rdf:type } w \texttt{ .}$$

entail the triple

$$u \texttt{ rdf:type } v \texttt{ .}$$

The pD* semantics for `someValuesFrom` may not be sufficient for various applications. However, it is possible to use RDF's blank nodes to capture OWL's complete iff condition for `someValuesFrom`, by combining rule rdfp15 with a second entailment rule for `someValuesFrom`. Rule rdf-svx states that three triples of the form

$$v \texttt{ owl:someValuesFrom } w \texttt{ .} \tag{13}$$

v `owl:onProperty` p . (14)

u `rdf:type` v . (15)

entail the two triples

u p b . (16)

b `rdf:type` w . (17)

Here b is a new blank node. It should be noted that the decidability and complexity results obtained in this paper for pD* entailment do not incorporate the use of rule rdf-svx. We do, however, extend the completeness result for pD* entailment to an extended entailment relation, called pD*sv entailment, which incorporates OWL's complete iff condition for `someValuesFrom` as well as entailment rule rdf-svx. In some situations the complete iff condition for `someValuesFrom` may already be available in combination with a reasoner that supports only the pD* semantics. Suppose that a person or an application wants to add the statement that a certain item u belongs to the class v, where v is the `someValuesFrom` restriction defined by triples (13) and (14). Then, instead of adding the triple (15), the triples (16) and (17) can be added, where b is a new blank node. This pair of triples reflects exactly the intention of the class v, as u is declared to be related with respect to p to some instance of w, while nothing more is expressed about this instance of w. Under the pD* semantics, the two triples (16) and (17) are stronger than the triple (15), since the latter triple is derived by means of rule rdfp15. This more controlled way of introducing new blank nodes may be useful in some situations in combination with the use of a reasoner that supports only the pD* semantics.

Although the pD* semantics does not handle explicitly `owl:unionOf` and `owl:intersectionOf`, half of OWL's iff conditions for these constructs are available in an alternative way, by means of `rdfs:subClassOf`. It can be expressed that the union of the classes $c_1, \ldots, c_n$ is contained in a class c by saying that each class c_j is a subclass of c. Moreover, it can be expressed that a class c is contained in the intersection of the classes $c_1, \ldots, c_n$ by saying that c is a subclass of each class c_j.

The pD* semantics does not consider `owl:complementOf`. However, part of the semantics of OWL for this construct can be obtained by using the following meta-modeling triple, which is consistent with the semantics of OWL:

`owl:complementOf rdfs:subPropertyOf owl:disjointWith .` (18)

Consistency with respect to the pD* semantics can be checked in polynomial time. An RDF graph is inconsistent with respect to the pD* semantics if and only if a partial closure of this graph contains a D-clash (see 1.7) or a P-clash; a P-clash is either a combination of two triples of the form

v `owl:differentFrom` w .
v `owl:sameAs` w .

or a combination of three triples of the form

v `owl:disjointWith` w .
u `rdf:type` v .
u `rdf:type` w .

Why is it reasonable, apart from complexity considerations, to consider a weakened semantics for OWL? Which entailments are missed? Before considering these questions further for OWL, we briefly consider them for RDFS. In Sections 4.2 and 7.3.1 of [8] an alternative semantics for RDFS is described, which is specified by means of iff conditions and allows conclusions to be drawn about entire classes or properties (cf. 1.2 above). For example, under an iff-semantics for RDFS two triples of the form

p `rdfs:domain` v . (19)
v `rdfs:subClassOf` w . (20)

entail the triple

p `rdfs:domain` w . (21)

This entailment is not an RDFS entailment. However, entailments obtained when the entailed triple (21) would have been added to the triples (19) and (20) are already obtained under the standard if-semantics for RDFS given only the original two triples (19) and (20). For example, the two triples (19) and (20) plus the triple

a p b .

RDFS-entail, with or without the triple (21), the triple

a `rdf:type` w .

It turns out that entailment rules for the stronger iff-semantics for RDFS become more complex; no complete set of entailment rules has been described. Some preliminary versions of [8] used iff conditions for `rdfs:subClassOf` and `rdfs:subPropertyOf`; after an error was found in a preliminary version of the RDFS entailment lemma, these conditions were changed to if conditions.

The standard if-semantics for RDFS is "deliberately chosen to be the weakest 'reasonable' interpretation of the RDFS vocabulary" [8].

Just as for RDFS, certain reasonable entailments are missing from the pD* entailment relation. For example, under the iff conditions defining OWL, the triple

p `rdf:type owl:SymmetricProperty .`

entails the triple

p `owl:inverseOf` p .

Moreover, the triple

p `owl:inverseOf` q .

iff-entails the triple

q `owl:inverseOf` p .

To give another example of an iff-entailment, the two triples

p `rdf:type owl:FunctionalProperty .` (22)
p `owl:inverseOf` q . (23)

entail the triple

q `rdf:type owl:InverseFunctionalProperty .` (24)

Moreover, two triples of the form

p `rdfs:domain` u .
p `owl:inverseOf` q .

iff-entail the triple

q `rdfs:range` u .

As a final example, four triples of the form

p rdfs:range c .
v owl:allValuesFrom c .
v owl:onProperty p .
a rdf:type rdfs:Resource .

iff-entail the triple

a rdf:type v .

These iff-entailments are not pD* entailments. However, just as for the RDFS example described, in examples such as these the iff-entailed statements do not lead to new consequences when the combination is made with data relating to instances. It is not difficult to check this with the entailment rules (see Tables 4 and 7). As an example, the two triples (22) and (23) plus the two triples

u q w .
v q w .

pD* entail, with or without the triple (24), the triple

u owl:sameAs v .

The pD* semantics does not include semantic conditions for the complete OWL vocabulary. It leaves out, in particular, the cardinality restrictions and `oneOf`. We mentioned that part of OWL's semantics for `unionOf`, `intersectionOf` and `complementOf` can be obtained in an alternative way. Extension of the pD* semantics to deal with the part of the OWL vocabulary not considered explicitly will lead to some more complicated entailment rules, for example because OWL uses lists encoded by means of RDF triples.

We discussed that the pD* semantics misses certain reasonable entailments for the part of the OWL vocabulary it does consider. In particular, applications that use `someValuesFrom` may need OWL's stronger semantic conditions, which are also included in the pD*sv semantics. However, examples such as those discussed also suggest that an if-semantics seems to be sufficient for many applications where an ontology is used in combination with data relating to instances. To summarize examples such as these, iff-entailments missing from the pD* semantics make explicit certain statements (about classes and properties) which do not lead to new consequences when the combination is made with data relating to instances.

OWL DL imposes restrictions on use to obtain decidability; in particular, classes and properties cannot be used as instances, and the use of `Functional-`

`Property` and `TransitiveProperty` is restricted (see [14], Section 2.3.1.3 for the latter restrictions on the use of OWL DL and OWL Lite). The pD* semantics does not impose restrictions on use to obtain decidability and supports meta-modeling expressivity. In view of the complete set of simple entailment rules, and the result that pD* entailment is in NP, and in a non-trivial case even in P, it can be said that, just as for RDFS, an if-semantics applied to (a subset of) the OWL vocabulary leads to computational advantages.

2 RDF Graphs and Simple Entailment

An RDF or OWL knowledge base is formalized as an RDF graph. This section recalls basic concepts relating to RDF graphs and simple entailment [8,12], and introduces generalizations as well as some notation. This section also considers the interpolation lemma [8] which characterizes simple entailment (cf. 1.4) and proves NP-completeness of simple entailment. The basic tool that we use in the remainder of this paper is set theory [7]. As has already been mentioned in 1.5, in order to obtain a complete set of entailment rules for RDFS, we adopt a syntactic extension of RDF. We use *generalized RDF graphs*, allowing blank nodes in predicate position. Instead of only using generalized RDF graphs as closure graphs of ordinary RDF graphs, we also state and prove the results of this paper for generalized RDF graphs. This requires hardly any additional effort.

2.1 URI references, blank nodes, literals. We first recall various notions from [12,8], define the symbols U, B, L also used by other authors and add the symbol T. RDF distinguishes three sets of syntactic entities. Let U denote the set of *URI references* (i.e. Uniform Resource Identifiers) and B the set of *blank nodes*, i.e. variables. The set B is assumed to be infinite. Let L be the set of *literals*, i.e. data values such as strings and integers; L is the union of 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: we shall write l as a pair, $l = (s, t)$. The sets U, B, L_{p} and L_{t} are pairwise disjoint. A *name* is an element of $U \cup L$ and a *vocabulary* is a subset of $U \cup L$. RDF has a special datatype URI, called `rdf:XMLLiteral`, which will also be written as `XMLLiteral`. An *XML literal* is a typed literal of the form $(s, \texttt{XMLLiteral})$. XML literals enable pieces of XML content to be used as data values. In addition to these notions from [12,8], an *RDF term* is either a URI reference, a blank node or a literal, in line with the usage by [16]. The set of RDF terms will be denoted by T: $T = U \cup B \cup L$.

2.2 RDF graphs and generalized RDF graphs. An *(RDF) graph* G [12,8]

is defined to be a subset of the set

$$U \cup B \times U \times U \cup B \cup L\,. \tag{25}$$

A *generalized (RDF) graph* G is defined as a subset of the set

$$U \cup B \times U \cup B \times U \cup B \cup L\,. \tag{26}$$

The elements (s, p, o) of a (generalized) RDF graph are called *(generalized) RDF statements* or *(generalized) RDF triples*, which consist of a *subject*, a *predicate* (or *property*) and an *object*, respectively. We write triples as $s\,p\,o$ and use meta-variables for RDF terms in order to shorten definitions and proofs. The notation can be viewed as an abbreviation of the N-Triples notation [5] used in the examples in Section 1. [5]

Two simple definitions [8] extend immediately from RDF graphs to generalized RDF graphs: a *subgraph* of a generalized RDF graph is a subset of the graph, while a generalized graph is *ground* if it has no blank nodes.

It is useful to add some notation. Denoting the projection mappings on the three factor sets of the product set given in (26) by π_i, the *set of RDF terms* of a generalized RDF graph G will be denoted by

$$T(G) = \pi_1(G) \cup \pi_2(G) \cup \pi_3(G),$$

which is a subset of $U \cup B \cup L$. The set of *blank nodes* of a generalized RDF graph G will be denoted by

$$bl(G) = T(G) \cap B.$$

The *vocabulary of a generalized RDF graph* G, denoted by $V(G)$, is the set of names that occur as subject, predicate or object of a triple in G:

$$V(G) = T(G) \cap (U \cup L).$$

The *set of nodes* of a generalized RDF graph G is

$$nd(G) = \pi_1(G) \cup \pi_3(G).$$

[5] As in the expression $s\,p\,o \in G$, where G is an RDF graph, the context will always make clear what the triple is. In the proofs we sometimes express that two triples $s\,p\,o$ and $s'\,p'\,o'$ are equal (i.e. that $s = s'$, $p = p'$ and $o = o'$) by writing $s\,p\,o = s'\,p'\,o'$.

2.3 Equivalence. Two RDF graphs G and G' are *equivalent* [12] if there is a bijection $f : nd(G) \rightarrow nd(G')$ such that $f(bl(G)) \subseteq bl(G')$, such that $f(v) = v$ for each $v \in nd(G) \cap (U \cup L)$, and such that $s\,p\,o \in G$ if and only if $f(s)\,p\,f(o) \in G'$. This is the only definition in this paper for which the extension to generalized RDF graphs is not trivial. We extend this definition in the following way to generalized graphs. Two generalized RDF graphs G and G' are *equivalent* if there is a bijection $f : T(G) \rightarrow T(G')$ such that $f(bl(G)) \subseteq bl(G')$, such that $f(v) = v$ for each $v \in V(G)$, and such that $s\,p\,o \in G$ if and only if $f(s)\ f(p)\ f(o) \in G'$. Note that equivalence forms an equivalence relation on the class of all generalized RDF graphs.

The following two definitions provide straightforward extensions to generalized RDF graphs of standard definitions for RDF graphs [8].

2.4 Instance. Given a generalized RDF graph G and a partial function $h : B \rightharpoonup T$, another generalized RDF graph is defined, the *instance* G_h of G, obtained by replacing the blank nodes v that are both in G and the domain of h by $h(v)$.

2.5 Merge. Given a set S of generalized RDF graphs, a *merge* of S is a generalized RDF graph that is obtained by replacing the generalized graphs G in S by equivalent generalized graphs G' that do not share blank nodes, and by taking the union of these generalized graphs G'. The merge of a set of generalized RDF graphs S is uniquely defined up to equivalence. A merge of S will be denoted by $M(S)$.

2.6 Simple interpretations. A *simple interpretation* I [8] of a vocabulary V is a 6-tuple $I = (R_I, P_I, E_I, S_I, L_I, LV_I)$, where R_I is a non-empty set, called the set of *resources* or the *universe*, P_I is the set of *properties*, 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:

$$E_I :\ P_I \ \rightarrow\ \mathcal{P}(R_I \times R_I),$$

$$S_I :\ V \cap U \ \rightarrow\ R_I \cup P_I,$$

$$L_I :\ V \cap L_{\mathrm{t}} \ \rightarrow\ R_I.$$

Here $\mathcal{P}(X)$ denotes the power set of the set X, i.e. the set of all subsets of X. The function E_I defines the extension of a property as a set of pairs of resources. The functions S_I and L_I define the interpretation of URI references and typed literals, respectively.

If I is a simple interpretation of a vocabulary V, then I also denotes a function with domain V in the following way. For plain literals $l \in L_{\mathrm{p}} \cap V$, we have

$I(l) = l \in LV_I$. For typed literals $l \in L_{\mathrm{t}} \cap V$, $I(l) = L_I(l)$. For URI references $a \in U \cap V$, $I(a) = S_I(a)$.

If $E = s\,p\,o$ is a ground triple, then a simple interpretation I of a vocabulary V is said to *satisfy* E if $s, p, o \in V$, $I(p) \in P_I$, and $(I(s), I(o)) \in E_I(I(p))$. If G is a ground RDF graph, then I satisfies G if I satisfies each triple $E \in G$.

Given a simple interpretation I and a partial function $A : B \rightharpoonup R_I$, a function I_A is defined that extends I by using A to give an interpretation of blank nodes in the domain of A. If $A(v)$ is defined for $v \in B$, then $I_A(v) = A(v)$. If G is any generalized RDF graph, then I satisfies G if I_A satisfies G for some function $A : bl(G) \to R_I$, i.e. if, for each triple $s\,p\,o \in G$, we have $I_A(p) \in P_I$ and

$$(I_A(s), I_A(o)) \in E_I(I_A(p))\,.$$

If I is a simple interpretation and S a set of generalized RDF graphs, then I satisfies S if I satisfies G for each G in S; it is not difficult to see that I satisfies S if and only if I satisfies $M(S)$.

2.7 Simple entailment. The definition of simple entailment [8] is extended in a straightforward way to all generalized RDF graphs: the set S of generalized RDF graphs *simply entails* a generalized RDF graph G if each simple interpretation that satisfies S also satisfies G. In this case we write

$$S \models G.$$

For RDF graphs, simple entailment is characterized by the interpolation lemma proved in [8].

2.8 Interpolation lemma. *If S is a set of RDF graphs and G is an RDF graph, then $S \models G$ if and only if a subset of $M(S)$ is an instance of G.*

This result shows that the simple-entailment relation $S \models G$ between finite sets S of finite RDF graphs and finite RDF graphs G is decidable. We shall now show that the interpolation lemma extends to all generalized RDF graphs by adapting the proof of the interpolation lemma given in [8]. The proof contains in simple form several key ideas that will reappear in more complicated ways for RDFS and OWL in the following sections. We first need to extend two other lemmas from [8] to generalized RDF graphs.

2.9 Generalized instance lemma. *If H and G are generalized RDF graphs and if H is an instance of G, then $H \models G$.*

Proof. Suppose that the simple interpretation I satisfies H, and suppose that

H is obtained from G by the function $h : bl(G) \to T(H)$. Extend h to become a function $h : T(G) \to T(H)$, by means of $h(v) = v$ for each v in $T(G) - bl(G)$. Choose $A : bl(H) \to R_I$ such that I_A satisfies H. Let $s\,p\,o \in G$. Then $h(s)$ $h(p)$ $h(o) \in H$, so that $I_A(h(p)) \in P_I$ and

$$(I_A(h(s)), I_A(h(o))) \in E_I(I_A(p)).$$

Define $A' : bl(G) \to R_I$ to be

$$A'(b) = I_A(h(b)) \quad (b \in bl(G)) \ .$$

Then we have

$$I_{A'}(v) = I_A(h(v)) \quad (v \in T(G)) \ ,$$

so that

$$(I_{A'}(s), I_{A'}(o)) \in E_I(I_{A'}(p)).$$

Hence I satisfies G. □

2.10 Definition (Herbrand interpretation). The *Herbrand interpretation* of a non-empty generalized RDF graph G is an interpretation of the vocabulary $V(G)$, constructed by means of G itself. It is defined as follows, generalizing the notion of simple Herbrand interpretation from [8]:

$$R_I = T(G)$$

$$LV_I = T(G) \cap L_{\mathrm{p}}$$

$$P_I = \pi_2(G)$$

$$S_I(v) = v \ \ (v \in V(G) \cap U)$$

$$L_I(v) = v \ \ (v \in V(G) \cap L_t)$$

$$E_I(p) = \{(s, o) : \ s\,p\,o \in G\} \ \ (p \in P_I)$$

The Herbrand interpretation of a generalized RDF graph G will be denoted by $H(G)$.

2.11 Generalized satisfaction lemma. *Each generalized RDF graph G has a satisfying interpretation: if G is non-empty, then $H(G)$ satisfies G.*

Proof. The empty graph is satisfied by any interpretation. Suppose G is a non-empty generalized graph, let I be the Herbrand interpretation $H(G)$ of G, and let A be the identity function on $bl(G)$. Then $I_A(v) = v$ for each $v \in T(G)$, so $(I_A(s), I_A(o)) \in E_I(I_A(p))$ for each triple $s\,p\,o \in G$. Therefore, I satisfies G. □

2.12 Generalized interpolation lemma. *If S is a set of generalized RDF graphs and G a generalized RDF graph, then $S \models G$ if and only if there is a subset of $M(S)$ that is an instance of G.*

Proof. ⇒ If S is empty, then it is clear that G must be empty as well, so that the conclusion follows. If S is non-empty, let I be the Herbrand interpretation of $M(S)$: $I = H(M(S))$. (It is assumed that the set B of blank nodes is large enough to form the merge $M(S)$.) Since I satisfies S by the satisfaction lemma, I satisfies G. Choose $A : bl(G) \to R_I$ such that I_A satisfies G. It follows that for each triple $s\,p\,o \in G$, $M(S)$ contains the triple $I_A(s)\ I_A(p)\ I_A(o)$, which is the instance of $s\,p\,o$ under the instance mapping A. Hence, there is a subset of $M(S)$ that is an instance of G.

⇐ Suppose H is a subset of $M(S)$ and an instance of G. Then, the instance lemma shows that $H \models G$, and it is clear that $M(S) \models H$. Hence $M(S) \models G$ and $S \models G$. □

The (generalized) interpolation lemma not only shows that the simple-entailment relation $S \models G$ between finite sets S of finite (generalized) RDF graphs and finite (generalized) RDF graphs G is decidable, but also shows that this problem is in NP: guess an instance function $h : bl(G) \to T(M(S))$, and check that the instance G_h of G defined by means of this function is a subset of $M(S)$.

It is clear that this problem, the simple-entailment relation $S \models G$ between finite sets S of finite (generalized) RDF graphs and finite (generalized) RDF graphs G, is in P if G is assumed to be ground, for in this case it needs only to be checked that G is a subset of $M(S)$.

According to [8], the full simple-entailment problem (without restrictive assumptions) is NP-complete. It seems that no complete proof has been published yet. A proof is outlined in one sentence in [8]: "This can be shown by encoding the problem of detecting a subgraph of an arbitrary directed graph as an RDF entailment, using only blank nodes to represent the graph (observation due to Jeremy Carroll)." Note that it is not trivial to work out the details of this proof sketch to obtain a complete proof, as the definition of instance (see 2.4) does not require the instance functions h to be injective.

We give a full proof of the NP-completeness of simple entailment by reduction from another standard NP-complete problem [4], the clique problem. Although the NP-completeness results in this paper (see the following proposition and

Propositions 4.14 and 5.15) are stated for generalized RDF graphs, they clearly also hold for RDF graphs because their proofs use only RDF graphs.

2.13 Proposition. *The simple-entailment relation $S \models G$ between finite sets S of finite generalized RDF graphs and finite generalized RDF graphs G is NP-complete.*

Proof. As shown above, this problem is in NP. We prove NP-completeness using a reduction from the clique problem. An instance of the clique problem consists of a finite undirected graph $G = (V, E)$, where E can be assumed to contain no loops $\{v, v\}$, and a positive integer $k \leq |V|$. The clique problem asks whether G has a clique of size $\geq k$, i.e. whether there exists a set of nodes $W \subseteq V$ of size $|W| \geq k$, such that each pair of distinct nodes $v, w \in W$ is linked by an edge in G: $\{v, w\} \in E$. An instance $G = (V, E), k$ of the clique problem is transformed to RDF graphs G' and H'. Each of the triples in G' and H' has the same predicate p, and all nodes of G' and H' are blank nodes. The RDF graph G' is formed by converting each pair $\{v, w\} \in E$ into two triples $v\,p\,w$ and $w\,p\,v$. The RDF graph H' consists of the triples $v\,p\,w$ where v and w are distinct elements of an arbitrary set of exactly k blank nodes. It is clear that the transformation from G, k to G', H' can be done in polynomial time. We need to prove that the finite undirected graph without loops G has a clique of size $\geq k$ if and only if G' simply entails H'. In view of the interpolation lemma, it is sufficient to prove that the finite undirected graph without loops G has a clique of size $\geq k$ if and only if there is a function $h : bl(H') \to bl(G')$ satisfying $H'_h \subseteq G'$. The only if direction is clear. Furthermore, if there is such a function h, then h needs to be injective. Note that H' otherwise has distinct blank nodes v and w such that $h(v) = h(w)$. However, then it would follow that $h(v)\ p\ h(v) \in G'$, contradicting the assumption that G does not contain loops. Since h is injective, G has a clique of size $\geq k$. □

3 RDFS Interpretations and D* Interpretations

As preparation for the proof of decidability of entailment for RDFS, possibly extended with datatypes, this section defines D* interpretations, a generalization of RDFS interpretations extending the semantics of the datatype `XMLLiteral` to all datatypes in a datatype map. We first consider the notion of RDFS interpretation [8], which is defined here in a slightly more refined way than in [8], in order to show that RDFS interpretations are mathematically well defined.

3.1 RDFS interpretations. The *RDF* and *RDFS vocabulary*, $rdfV \cup rdfsV$, is a set of URI references listed in Table 1. Except for the container membership properties `rdf:_`i, we shall often omit the prefixes `rdf:` and `rdfs:` from

Table 1
RDF and RDFS URIs [8].

rdf:type	rdf:Seq	rdfs:Datatype
rdf:Property	rdf:Bag	rdfs:Class
rdf:XMLLiteral	rdf:Alt	rdfs:subClassOf
rdf:nil	rdf:_1	rdfs:subPropertyOf
rdf:List	rdf:_2	rdfs:member
rdf:Statement	...	rdfs:Container
rdf:subject	rdf:value	rdfs:ContainerMembershipProperty
rdf:predicate	rdfs:domain	rdfs:comment
rdf:object	rdfs:range	rdfs:seeAlso
rdf:first	rdfs:Resource	rdfs:isDefinedBy
rdf:rest	rdfs:Literal	rdfs:label

these URIs.

Let V be a vocabulary and I a simple interpretation of $V \cup \{\texttt{type}, \texttt{Class}\}$ such that $I(\texttt{type}) \in P_I$, so that $E_I(I(\texttt{type}))$ is defined. In this case, the set C_I of *classes* of I is defined to be

$$C_I = \{a \in R_I : (a, I(\texttt{Class})) \in E_I(I(\texttt{type}))\},$$

and the *class extension function* $CE_I : C_I \to \mathcal{P}(R_I)$ of I is defined by

$$CE_I(a) = \{b \in R_I : (b, a) \in E_I(I(\texttt{type}))\} \quad (a \in C_I).$$

An *rdfs-interpretation* of a vocabulary V is a simple interpretation I of $V \cup rdfV \cup rdfsV$ that satisfies the following conditions:

- I satisfies all triples in Tables 2 and 3. These triples are together called the *RDF* and *RDFS axiomatic triples*.
- $a \in P_I$ if and only if $(a, I(\texttt{Property})) \in E_I(I(\texttt{type}))$
- $a \in R_I$ if and only if $(a, I(\texttt{Resource})) \in E_I(I(\texttt{type}))$
- $a \in LV_I$ if and only if $(a, I(\texttt{Literal})) \in E_I(I(\texttt{type}))$
- If $(a, b) \in E_I(I(\texttt{domain}))$, $a \in P_I$, $b \in C_I$ and $(e, f) \in E_I(a)$, then $e \in CE_I(b)$ [6]
- If $(a, b) \in E_I(I(\texttt{range}))$, $a \in P_I$, $b \in C_I$ and $(e, f) \in E_I(a)$, then $f \in CE_I(b)$
- $E_I(I(\texttt{subPropertyOf}))$ is transitive and reflexive on P_I
- If $(a, b) \in E_I(I(\texttt{subPropertyOf}))$, then $a \in P_I$, $b \in P_I$ and $E_I(a) \subseteq E_I(b)$
- If $a \in C_I$ then $(a, I(\texttt{Resource})) \in E_I(I(\texttt{subClassOf}))$

[6] This condition and the next condition have been reformulated slightly in comparison with [8] to make clear that the definition uses the functions E_I and CE_I inside their domains, as required. See the explanation following the definition. Lemma 3.2 below shows that exactly the same class of rdfs-interpretations is defined as in [8].

Table 2
RDF axiomatic triples [8].

`type type Property .`	`value type Property .`
`subject type Property .`	`rdf:_1 type Property .`
`predicate type Property .`	`rdf:_2 type Property .`
`object type Property .`	`...`
`first type Property .`	`nil type List .`
`rest type Property .`	

Table 3
RDFS axiomatic triples [8].

`type domain Resource .`	`first range Resource .`
`domain domain Property .`	`rest range List .`
`range domain Property .`	`seeAlso range Resource .`
`subPropertyOf domain Property .`	`isDefinedBy range Resource .`
`subClassOf domain Class .`	`comment range Literal .`
`subject domain Statement .`	`label range Literal .`
`predicate domain Statement .`	`value range Resource .`
`object domain Statement .`	`Alt subClassOf Container .`
`member domain Resource .`	`Bag subClassOf Container .`
`first domain List .`	`Seq subClassOf Container .`
`rest domain List .`	`ContainerMembershipProperty`
`seeAlso domain Resource .`	`subClassOf Property .`
`isDefinedBy domain Resource .`	`isDefinedBy subPropertyOf seeAlso .`
`comment domain Resource .`	`XMLLiteral type Datatype .`
`label domain Resource .`	`XMLLiteral subClassOf Literal .`
`value domain Resource .`	`Datatype subClassOf Class .`
`type range Class .`	`rdf:_1 type`
`domain range Class .`	`ContainerMembershipProperty .`
`range range Class .`	`rdf:_1 domain Resource .`
`subPropertyOf range Property .`	`rdf:_1 range Resource .`
`subClassOf range Class .`	`rdf:_2 type`
`subject range Resource .`	`ContainerMembershipProperty .`
`predicate range Resource .`	`rdf:_2 domain Resource .`
`object range Resource .`	`rdf:_2 range Resource .`
`member range Resource .`	`...`

- If $(a, b) \in E_I(I(\texttt{subClassOf}))$, then $a \in C_I$, $b \in C_I$ and $CE_I(a) \subseteq CE_I(b)$
- $E_I(I(\texttt{subClassOf}))$ is transitive and reflexive on C_I
- If $a \in CE_I(I(\texttt{ContainerMembershipProperty}))$, then $(a, I(\texttt{member})) \in E_I(I(\texttt{subPropertyOf}))$
- If $a \in CE_I(I(\texttt{Datatype}))$, then $(a, I(\texttt{Literal})) \in E_I(I(\texttt{subClassOf}))$
- If $l = (s, \texttt{XMLLiteral}) \in V$ is a well-typed XML literal, then $L_I(l) = xml(l) \in LV_I$ and $(L_I(l), I(\texttt{XMLLiteral})) \in E_I(I(\texttt{type}))$
- If $l = (s, \texttt{XMLLiteral}) \in V$ is an XML literal that is not well typed, then $L_I(l) \notin LV_I$ and $(L_I(l), I(\texttt{XMLLiteral})) \notin E_I(I(\texttt{type}))$

Here the function xml assigns to each well-typed XML literal its value [12].

Note that the first axiomatic triple, `type type Property`, shows that for each rdfs-interpretation I, $I(\texttt{type}) \in P_I$, so that $E_I(I(\texttt{type}))$, C_I and CE_I are defined, as used in the statement of the remaining conditions. By using the functions E_I and CE_I, the definition assumes, implicitly, that certain values of I, such as $I(\texttt{domain})$ and $I(\texttt{range})$, are in P_I and that other values of I, such as $I(\texttt{Datatype})$, are in C_I. That this is allowed is demonstrated in the next lemma. This lemma also justifies some other conclusions that can be stated for each rdfs-interpretation and that are used constantly in the manipulation of rdfs-interpretations.

3.2 Lemma. *rdfs-interpretations are well defined, in the sense that each invocation of the functions E_I and CE_I in the definition is allowed in view of the earlier conditions. If I is an rdfs-interpretation of a vocabulary V, then*

- $I(\texttt{type})$, $I(\texttt{domain})$, $I(\texttt{range})$, $I(\texttt{subClassOf})$, $I(\texttt{subPropertyOf})$ *and* $I(\texttt{member})$ *are in* P_I
- $I(\texttt{Property})$, $I(\texttt{Resource})$, $I(\texttt{Class})$, $I(\texttt{Literal})$, $I(\texttt{Datatype})$, $I(\texttt{XMLLiteral})$ *and* $I(\texttt{ContainerMembershipProperty})$ *are in* C_I
- $P_I \subseteq R_I$
- $P_I = CE_I(I(\texttt{Property}))$
- $C_I = CE_I(I(\texttt{Class}))$
- $R_I = CE_I(I(\texttt{Resource}))$
- $LV_I = CE_I(I(\texttt{Literal}))$
- *if* $(a, b) \in E_I(I(\texttt{type}))$, *then* $b \in C_I$ *and* $a \in CE_I(b)$
- *if* (a, b) *is in* $E_I(I(\texttt{domain}))$ *or* $E_I(I(\texttt{range}))$, *then* $a \in P_I$ *and* $b \in C_I$
- *if* $(a, b) \in E_I(I(\texttt{domain}))$ *and* $(e, f) \in E_I(a)$, *then* $e \in CE_I(b)$
- *if* $(a, b) \in E_I(I(\texttt{range}))$ *and* $(e, f) \in E_I(a)$, *then* $f \in CE_I(b)$

Proof. Suppose first that I is a simple interpretation of $V \cup \mathit{rdfV} \cup \mathit{rdfsV}$ that satisfies all RDF and RDFS axiomatic triples. It has already been noted that $I(\texttt{type}) \in P_I$. If $p =$ `domain`, `range`, `subClassOf` or `subPropertyOf`, then I satisfies an axiomatic triple $v\,p\,w$ with p in property position (see Table 3), so that $I(\texttt{domain})$, $I(\texttt{range})$, $I(\texttt{subClassOf})$ and $I(\texttt{subPropertyOf})$ are in P_I. If I satisfies the second condition on `subClassOf` from the definition of rdfs-interpretations, then since I satisfies the triples `ContainerMembershipProperty subClassOf Property`, `Datatype subClassOf Class` and `XMLLiteral subClassOf Literal`, it follows that $I(\texttt{Property})$, $I(\texttt{Class})$, $I(\texttt{Literal})$, $I(\texttt{Datatype})$, $I(\texttt{XMLLiteral})$ and $I(\texttt{ContainerMembershipProperty})$ are in C_I. This proves the first statement and most of the first two conclusions listed.

Suppose next that I is an rdfs-interpretation of V. The condition that $a \in P_I$ if and only if $(a, I(\texttt{Property})) \in E_I(I(\texttt{type}))$ implies that $P_I \subseteq R_I$. Since $I(\texttt{Property}) \in C_I$, the first semantic condition on `subClassOf` shows that $(I(\texttt{Property}), I(\texttt{Resource})) \in E_I(I(\texttt{subClassOf}))$, so that, by the second

condition on `subClassOf`, it follows that $I(\texttt{Resource}) \in C_I$. It is now clear that $P_I = CE_I(I(\texttt{Property}))$, $C_I = CE_I(I(\texttt{Class}))$, $R_I = CE_I(I(\texttt{Resource}))$ and $LV_I = CE_I(I(\texttt{Literal}))$. It follows that if $(a, b) \in E_I(I(\texttt{type}))$, then $b \in C_I$ and $a \in CE_I(b)$: the semantic condition on $I(\texttt{range})$, the axiomatic triple `type range Class` and $(a, b) \in E_I(I(\texttt{type}))$ imply that $b \in CE_I(I(\texttt{Class})) = C_I$, and then it also follows that $a \in CE_I(b)$.

It remains only to be shown that if (a, b) is in $E_I(I(\texttt{domain}))$ or $E_I(I(\texttt{range}))$, then $a \in P_I$ and $b \in C_I$. If this is proved, then the last two statements to be proved follow immediately, and the fact that I satisfies the axiomatic triple `member domain Resource` implies that $I(\texttt{member}) \in P_I$. So suppose that $(a, b) \in E_I(I(\texttt{domain}))$. Since I satisfies the triple `domain domain Property`, we get $(I(\texttt{domain}), I(\texttt{Property})) \in E_I(I(\texttt{domain}))$. Combining this with $I(\texttt{domain}) \in P_I$ and $I(\texttt{Property}) \in C_I$, the semantic condition on $I(\texttt{domain})$ shows that $a \in CE_I(I(\texttt{Property})) = P_I$. Since I satisfies `domain range Class`, $(I(\texttt{domain}), I(\texttt{Class})) \in E_I(I(\texttt{range}))$. Since $I(\texttt{domain}) \in P_I$ and $I(\texttt{Class}) \in C_I$, we get $b \in CE_I(I(\texttt{Class})) = C_I$. The proof for $I(\texttt{range})$ follows in the same way. □

3.3 Datatype maps. Before defining D* interpretations, we summarize some terminology and introduce some notation relating to datatype maps [12,8]. Intuitively, a datatype is characterized by certain strings (e.g. `"11"`) and corresponding values (e.g. the number 11). Formally, a *datatype* d is defined by a non-empty set of strings $L(d)$, the *lexical space*, a non-empty set $V(d)$, the *value space*, and a function $L2V(d) : L(d) \to V(d)$, the *lexical-to-value mapping*. A *datatype map* is a partial function D from the set U of URI references to the class of datatypes. Each datatype map is required [8] to contain the pair $(\texttt{XMLLiteral}, x)$, where x is the built-in XML literal datatype, defined by $L(x) = \{s : (s, \texttt{XMLLiteral}) \in L^+_X\}$, $V(x) = XV$, and $L2V(x)(s) = xml((s, \texttt{XMLLiteral}))$ for each $(s, \texttt{XMLLiteral}) \in L^+_X$. Here L^+_X is the set of well-typed XML literals [12].

It is useful to add some further notation relating to datatype maps. Suppose that a datatype map D is given. The *D-vocabulary* is the *domain* $dom(D)$ of D, i.e. the set of URI references $a \in U$ such that $(a, d) \in D$ for some datatype d. The *range* of D, i.e. the set of datatypes d such that $(a, d) \in D$ for some $a \in U$, is denoted by $ran(D)$. The set L_D of *D-literals* is the set of typed literals $(s, a) \in L_{\text{t}}$ with $a \in dom(D)$. The set of all well-typed literals with type in D, i.e. the set of all *well-typed D-literals*, will be denoted by L^+_D:

$$L^+_D = \{(s, a) \in L_{\text{t}} : a \in dom(D), s \in L(D(a))\}.$$

The function val_D is defined to map each well-typed D-literal to its value:

$$val_D : L_D^+ \to \bigcup_{d \in ran(D)} V(d) \ ,$$

$$val_D((s,a)) = L2V(D(a))(s) \quad (a \in dom(D), s \in L(D(a))) \ .$$

We assume that a few basic operations with respect to datatype maps can be executed in polynomial time; these operations include, in particular, the operation determining whether a given D-literal is well typed, the computation of values of well-typed D-literals, and the operation determining whether the values of two well-typed D-literals are equal.

3.4 Definition (D* interpretations). If D is a datatype map, a *D^* interpretation* of a vocabulary V is defined to be an rdfs-interpretation I of $V \cup dom(D)$ that satisfies the following conditions for each pair $(a,d) \in D$:

- $I(a) = d$
- I satisfies the following triples: a `type Datatype` and a `subClassOf Literal`
- if $l = (s,a) \in L_{\mathrm{t}} \cap V$ and $s \in L(d)$, then $L_I(l) = L2V(d)(s) \in LV_I$ and $(L_I(l), d) \in E_I(I(\texttt{type}))$
- if $l = (s,a) \in L_{\mathrm{t}} \cap V$ and $s \notin L(d)$, then $L_I(l) \notin LV_I$ and $(L_I(l), d) \notin E_I(I(\texttt{type}))$

3.5 Definition (D-axiomatic triples). If D is a datatype map, the triples a `type Datatype` and a `subClassOf Literal` appearing in the definition of D* interpretations are combined for $a \in dom(D)$ to form the *D-axiomatic triples*.

It should be noted that D* interpretations generalize the `XMLLiteral`-related conditions on rdfs-interpretations.[7] We conclude this section by showing that D-interpretations [8] form a special kind of D* interpretation. See 1.7 for an example showing that the D-semantics is stronger than the D* semantics.

3.6 Definition (D-interpretations). If D is a datatype map, a *D-interpretation* of a vocabulary V is an rdfs-interpretation I of $V \cup dom(D)$ that satisfies the following conditions for each pair $(a,d) \in D$:

- $I(a) = d \in CE_I(I(\texttt{Datatype}))$
- $CE_I(d) = V(d) \subseteq LV_I$
- if $l = (s,a') \in L_{\mathrm{t}} \cap V$, $I(a') = d$ and $s \in L(d)$, then $L_I(l) = L2V(d)(s)$

[7] The definition of D* interpretations has been slightly simplified with respect to the definition given in [10] in order to make this correspondence with the `XMLLiteral`-related conditions exact. As was already noted in [10] (footnote 4), this change does not lead to a change in entailments.

- if $l = (s, a') \in L_{\mathrm{t}} \cap V$, $I(a') = d$ and $s \notin L(d)$, then $L_I(l) \notin LV_I$

Note that D-interpretations are well defined, i.e. that $d \in C_I$: since $I(a) = d \in CE_I(I(\texttt{Datatype}))$, a condition on rdfs-interpretations shows that $(d, I(\texttt{Literal})) \in E_I(I(\texttt{subClassOf}))$, so that by another condition on rdfs-interpretations, $d \in C_I$.

3.7 Lemma. *If I is a D-interpretation of a vocabulary V, then I is a D* interpretation of V.*

Proof. Let $(a, d) \in D$. Since $I(a) = d \in CE_I(I(\texttt{Datatype}))$, I satisfies the triple a `type Datatype`. A condition on rdfs-interpretations shows that $(d, I(\texttt{Literal})) \in E_I(I(\texttt{subClassOf}))$, so that I satisfies the triple a `subClassOf Literal` as well. If $l = (s, a) \in L_{\mathrm{t}} \cap V$ and $s \in L(d)$, then $L_I(l) = L2V(d)(s) \in V(d) = CE_I(d)$, so that $L_I(l) \in LV_I$ and $(L_I(l), d) \in E_I(I(\texttt{type}))$. If $l = (s, a) \in L_{\mathrm{t}} \cap V$ and $s \notin L(d)$, then $L_I(l) \notin LV_I$, so that $(L_I(l), d) \notin E_I(I(\texttt{type}))$: otherwise, $L_I(l) \in CE_I(d) \subseteq LV_I$. □

4 RDFS Entailment and D* Entailment

Using the semantic definitions of the preceding section, this section defines RDFS entailment and D* entailment and turns to completeness, decidability and computational complexity. As was also done in earlier versions of [8], we extract from the completeness proof the notions of closure and Herbrand interpretation, as well as a satisfaction lemma (compare the proof of Lemma 2.12). Instead of a full closure, a partial closure is used which can be finite, making the closure notion of practical interest and no longer just an 'abstraction in the proof'. The Herbrand construction of [8] needs to be refined to deal with the finiteness of a partial closure and with datatypes.

4.1 Definition (D* entailment). Given a datatype map D, a set S of generalized RDF graphs *D* entails* a generalized RDF graph G if each D* interpretation I that satisfies S also satisfies G.

RDFS entailment coincides with D* entailment if the datatype map D is assumed to consist of only the type `rdf:XMLLiteral`. We use the notation

$$S \models_{\mathrm{s}} G$$

for D* entailment (and also for the special case of RDFS entailment).

Since each D-interpretation is a D* interpretation by Lemma 3.7, it is clear that if S D* entails G, then S D-entails G. Here we extend the definition of

D-entailment [8] to generalized RDF graphs in the following way.

4.2 Definition (D-entailment). Given a datatype map D, a set S of generalized RDF graphs *D-entails* a generalized RDF graph G if each D-interpretation I that satisfies S also satisfies G.

4.3 An example. Before turning to syntactic entailment rules for D* entailment, we note that it is possible to use the semantic definitions to give a direct proof of the correctness of an entailment. To illustrate this point, we prove that the example given in 1.5 is indeed an RDFS entailment. Suppose that I is an rdfs-interpretation that satisfies the triples (1), (2) and (3), so that there is a function A satisfying $(I_A(p), I_A(b)) \in E_I(I(\texttt{subPropertyOf}))$, $(I_A(b), I_A(u)) \in E_I(I(\texttt{domain}))$, $I_A(p) \in P_I$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$. Then a semantic condition on `subPropertyOf` shows that $I_A(b) \in P_I$ and $(I_A(v), I_A(w)) \in E_I(I_A(b))$, so that by Lemma 3.2 we have $I_A(v) \in CE_I(I_A(u))$. Therefore, $(I_A(v), I_A(u)) \in E_I(I(\texttt{type}))$, showing that I indeed satisfies the triple (4).

4.4 Definition (D* entailment rules). Given a datatype map D, the D* entailment rules are defined in Table 4. They consist of the 18 rules defined in [8] for RDFS, with two differences involving rules rdf2 and rdfs7. Instead of rule rdfs7 we use rule rdfs7x to solve the problem that the standard set of entailment rules for RDFS is incomplete (cf. 1.5). Rule rdfs7x differs only from rule rdfs7 in that it can produce generalized RDF triples with blank nodes in predicate position (see 2.2) when applied to ordinary RDF triples; rule rdfs7x is the only D* entailment rule with this property. While rule rdfs7 assumes that $q \in U$, rule rdfs7x assumes only that $q \in U \cup B$ (see Table 4). In order to deal with datatypes, rule rdf2 is replaced by the more general rule rdf2-D. The four D* entailment rules lg, gl, rdf2-D and rdfs1 involve certain blank nodes b_l, called *surrogate blank nodes*, which we now discuss in more detail. The first rule lg ('literal generalization') prescribes that if G contains $v\,p\,l$, where l is a literal, then add $v\,p\,b_l$ to G, where b_l is a blank node allocated to l by this rule. Here 'allocated to' means that the blank node b_l has been created by an application of rule lg on the same literal l, or if there is no such blank node, in which case it must be a new blank node which is not yet in the graph. In rule rdfs1, b_l is a blank node that is allocated by rule lg to the plain literal $l \in L_{\mathrm{p}}$. In rule rdf2-D, b_l is a blank node that is allocated by rule lg to the well-typed D-literal $l \in L_D^+$. Rule rdf2-D is a direct generalization of entailment rule rdf2 from [8], which has the same effect only for well-typed XML literals $l \in L_X^+$. If D contains only the datatype `rdf:XMLLiteral`, then rule rdf2-D becomes exactly rule rdf2.

4.5 Definition (D-clash). The notion of XML clash [8] is generalized in a straightforward way to any datatype map: given a datatype map D, a *D-clash* is a triple $b\,\texttt{type}\,\texttt{Literal}$, where b is a blank node allocated by rule lg to an ill-typed D-literal.

Table 4
D* entailment rules.

	If G contains	where	then add to G
lg	$v\,p\,l$	$l \in L$	$v\,p\,b_l$
gl	$v\,p\,b_l$	$l \in L$	$v\,p\,l$
rdf1	$v\,p\,w$		p `type` `Property`
rdf2-D	$v\,p\,l$	$l = (s,a) \in L_D^+$	b_l `type` a
rdfs1	$v\,p\,l$	$l \in L_{\mathrm{p}}$	b_l `type` `Literal`
rdfs2	p `domain` u		
	$v\,p\,w$		v `type` u
rdfs3	p `range` u		
	$v\,p\,w$	$w \in U \cup B$	w `type` u
rdfs4a	$v\,p\,w$		v `type` `Resource`
rdfs4b	$v\,p\,w$	$w \in U \cup B$	w `type` `Resource`
rdfs5	v `subPropertyOf` w		
	w `subPropertyOf` u		v `subPropertyOf` u
rdfs6	v `type` `Property`		v `subPropertyOf` v
rdfs7x	p `subPropertyOf` q		
	$v\,p\,w$	$q \in U \cup B$	$v\,q\,w$
rdfs8	v `type` `Class`		v `subClassOf` `Resource`
rdfs9	v `subClassOf` w		
	u `type` v		u `type` w
rdfs10	v `type` `Class`		v `subClassOf` v
rdfs11	v `subClassOf` w		
	w `subClassOf` u		v `subClassOf` u
rdfs12	v `type` `Container-`		
	`MembershipProperty`		v `subPropertyOf` `member`
rdfs13	v `type` `Datatype`		v `subClassOf` `Literal`

4.6 Definition (partial and full RDFS and D* closures). We turn to a possibly finite refinement of the notion of RDFS closure defined in [8]. Suppose that G is a generalized RDF graph and D a datatype map. In the definitions that follow, the axiomatic triples containing the URI references `rdf:_`i (i.e. the four triples `rdf:_`i `type` `Property`, `rdf:_`i `type` `ContainerMembership-Property`, `rdf:_`i `domain` `Resource` and `rdf:_`i `range` `Resource`) are treated in a special way. Suppose that K is a non-empty subset of the positive integers $\{1, 2, ...\}$ chosen in such a way that for each `rdf:_`$i \in V(G)$ we have $i \in K$. The *partial D* closure* $G_{\mathrm{s},K}$ of G is defined in the following way. In the first step, all RDF and RDFS axiomatic triples and D-axiomatic triples are added to G, except for the axiomatic triples including `rdf:_`i such that $i \notin K$. In the next step, rule lg is applied to each triple containing a literal in such a way that distinct, well-typed D-literals l with the same value are associated with the same surrogate blank node b_l. Then, rules rdf2-D and rdfs1 are applied to each triple containing a well-typed D-literal or a plain literal, respectively. Finally, arbitrary derivations (see Definition 4.7) are made using applications of the remaining D* entailment rules until the graph is unchanged. In addition to the partial D* closure $G_{\mathrm{s},K}$ obtained in this way, the *full D* closure* G_{s} of

G is defined by taking $G_{\mathrm{s}} = G_{\mathrm{s},K}$ where K is the full set $\{1, 2, ...\}$. If the datatype map D consists of only the datatype `rdf:XMLLiteral`, a partial and a full D* closure of a generalized RDF graph G are called a *partial* and a *full RDFS closure* of G, respectively.

As has already been mentioned, rule rdfs7x is the only D* entailment rule that may produce generalized RDF triples with blank nodes in predicate position when applied to ordinary RDF triples. In general, a partial D* closure is a generalized RDF graph, even if the given graph is an ordinary RDF graph. With respect to a partial (or full) D* closure $G_{\mathrm{s},K}$ and a literal $l \in V(G) \cap L$, the unique blank node allocated by rule lg to l is denoted by b_l.

4.7 Definition (derivation). Given a generalized RDF graph G, a set of entailment rules Γ and a generalized RDF triple E, a *derivation* of E from G by means of Γ is a finite sequence of RDF triples $E_1, ..., E_n$, such that $E_n = E$ and such that for each $i \leq n$, either $E_i \in G$ or E_i is obtained with a rule from Γ applied to one or more triples E_j such that $j < i$.

4.8 Lemma. *Let D be a finite datatype map. If G is a finite generalized RDF graph, then each partial D* closure $G_{\mathrm{s},K}$ of G is finite for K finite, and a partial D* closure of G can be computed in polynomial time.*

Proof. The graph obtained from a finite generalized RDF graph G in the first step of the definition of partial closure is clearly finite if K is finite. Then, rule lg adds only finitely many new triples, leading to a finite graph G' containing G. In the remaining steps, no new names or blank nodes are added to G', so it follows that there exist finite sets $U_0 \subseteq U$, $B_0 \subseteq B$ and $L_0 \subseteq L$ such that

$$G_{\mathrm{s},K} \subseteq (U_0 \cup B_0) \times (U_0 \cup B_0) \times (U_0 \cup B_0 \cup L_0)\,. \tag{27}$$

Hence $G_{\mathrm{s},K}$ is a finite graph.

To prove that a partial D* closure can be computed in polynomial time, choose a finite, non-empty set $K \subset \{1, 2, ...\}$ such that $i \in K$ for each `rdf:_`$i \in V(G)$. Let $g = |G|$, $d = |D|$ and $k = |K|$. We prove that $G_{\mathrm{s},K}$ can be computed in time polynomial in g, d and k. It will follow that a partial D* closure can be computed in time polynomial in g, since k can clearly be taken to be $3g+1$, and d can be viewed as a constant. The first step of the closure construction process adds $48 + 4k$ RDF and RDFS axiomatic triples and $2d$ D-axiomatic triples. In the next step, rule lg adds at most g triples. The computations needed for these applications of rule lg can be done in polynomial time, since we assumed that the involved basic operations connected to datatype maps (in particular, the operation to determine whether two well-typed D-literals have the same value) can be done in polynomial time. It follows that the graph G' that results from these steps has at most $2g + 2d + 4k + 48$ triples, so that the sets U_0, B_0

and L_0 in (27) can be chosen to satisfy $|U_0 \cup B_0 \cup L_0| \leq 3(2g + 2d + 4k + 48)$. This shows that $|G_{\mathrm{s},K}| \leq 27(2g + 2d + 4k + 48)^3$. Therefore, in the remaining steps at most $27(2g + 2d + 4k + 48)^3$ rule applications can add a new triple to the partial closure graph under construction. For each of the entailment rules used it can be determined whether a successful rule application exists in at most linear or quadratic time as a function of the size of the partial closure graph under construction (cf. Table 4). For example, for rule rdfs2 quadratic time is sufficient, while linear time is sufficient for rule rdf1. It follows that $G_{\mathrm{s},K}$ can be computed in time polynomial in g, d and k. □

4.9 Definition (D* Herbrand interpretation). Suppose that D is a datatype map. A *D* Herbrand interpretation* of a generalized RDF graph G is a simple interpretation $S_K(G)$ of $V(G_{\mathrm{s}})$ defined in the following way, refining the notion of rdfs-Herbrand interpretation defined in [8]. K is a non-empty subset of $\{1, 2, ...\}$ chosen such that for each $\texttt{rdf:\_}i \in V(G)$ we have $i \in K$. R_I is defined as the union of five pairwise disjoint sets:

$$val_D(V(G_{\mathrm{s}}) \cap L_D^+)$$

$$V(G_{\mathrm{s}}) \cap (L - L_D^+)$$

$$V(G_{\mathrm{s}}) \cap (U - dom(D))$$

$$ran(D)$$

$$bl(G_s)$$

The remainder of the definition makes use of the function

$$sur : R_I \to T(G_{\mathrm{s},K}) \cap (U \cup B)$$

('surrogate'), which replaces the elements of R_I with non-literal terms in $G_{\mathrm{s},K}$. This function is defined as follows:

- If $l \in V(G_{\mathrm{s}}) \cap L_D^+$, then $sur(val_D(l)) = b_l$.
- If $l \in V(G_{\mathrm{s}}) \cap (L - L_D^+)$, then $sur(l) = b_l$.
- If $v \in V(G_{\mathrm{s}}) \cap (U - (dom(D) \cup \{\texttt{rdf:\_}i : i \notin K\}))$, then $sur(v) = v$.
- If $i \notin K$, then $sur(\texttt{rdf:\_}i) = \texttt{rdf:\_}n$, where $n \in K$ is fixed.
- If $(a, d) \in D$, then $sur(d) = a$.
- If $b \in bl(G_s)$, then $sur(b) = b$.

Note that the *sur* function is well defined with respect to well-typed D-literals $l \in L_D^+$, since different well-typed D-literals $l \in V(G)$ with the same value $val_D(l)$ have the same surrogate blank node b_l. The other parts of $S_K(G)$ are

defined in the following way:

$$LV_I = \{x \in R_I : sur(x)\,\texttt{type}\,\texttt{Literal} \in G_{\mathrm{s},K}\}$$

$$P_I = \{x \in R_I : sur(x)\,\texttt{type}\,\texttt{Property} \in G_{\mathrm{s},K}\}$$

$$S_I(v) = v \quad (v \in V(G_\mathrm{s}) \cap U - dom(D))$$

$$S_I(a) = D(a) \quad (a \in V(G_\mathrm{s}) \cap dom(D))$$

$$L_I(v) = v \quad (v \in V(G_\mathrm{s}) \cap L_\mathrm{t} - L_D^+)$$

$$L_I(l) = val_D(l) \;\; (l \in V(G_\mathrm{s}) \cap L_D^+)$$

$$E_I(p) = \{(s,o) : s \in R_I, o \in R_I, sur(s)\,sur(p)\,sur(o) \in G_{\mathrm{s},K}\} \quad (p \in P_I)$$

Note that for each $v \in V(G_\mathrm{s}) \cap (U - \{\texttt{rdf:\_}i : i \notin K\})$, we have $sur(I(v)) = v$, and that $sur(I(\texttt{rdf:\_}i)) = \texttt{rdf:\_}n$ for $i \notin K$. Moreover, if $l \in V(G_\mathrm{s}) \cap L$, then $sur(I(l)) = b_l$.

4.10 D* satisfaction lemma. *Let G be a generalized RDF graph and D a datatype map. If the partial D* closure $G_{\mathrm{s},K}$ of G does not contain a D-clash, then $S_K(G)$ is a D* interpretation that satisfies $G_{\mathrm{s},K}$.*

Proof. In the proof the D* Herbrand interpretation $S_K(G)$ of G is denoted by I and the partial D* closure $G_{\mathrm{s},K}$ of G by H. The first thing that needs to be proved is that I is a simple interpretation, and for this we need to show that $V(G_\mathrm{s}) \cap L_\mathrm{p} \subseteq LV_I$. If $l \in V(G_\mathrm{s}) \cap L_\mathrm{p}$, then G contains a triple $v\,p\,l$. By rules lg and rdfs1, H contains triples $v\ p\ b_l$ and b_l $\texttt{type}$ $\texttt{Literal}$. Moreover, $sur(l) = b_l$, so that $sur(l)$ $\texttt{type}$ $\texttt{Literal} \in H$ and $l \in LV_I$. It is clear that if $v \in V(G_\mathrm{s}) \cap U$ then $S_I(v) \in R_I \cup P_I = R_I$, and that if $v \in V(G_\mathrm{s}) \cap L_\mathrm{t}$ then $L_I(v) \in R_I$.

Define the function $A : bl(H) \to R_I$ by $A(b_l) = val_D(l)$ if $l \in L_D^+$, $A(b_l) = l$ if $l \in L - L_D^+$, and by $A(b) = b$ for other blank nodes b. Note that the function A is well defined with respect to well-typed D-literals $l \in L_D^+$, since different well-typed D-literals $l \in V(G)$ with the same surrogate blank node b_l have the same value $val_D(l)$. Note, moreover, that

$$sur(I_A(v)) = v \quad (v \in T(H) - L)\,. \tag{28}$$

This can be seen easily in each of the following cases where $v \in T(H) - L$: $v \in U$, $v \in B$ allocated to a literal in L_D^+ or in $L - L_D^+$, other blank nodes v.

We prove that I_A satisfies H. Rule rdf1 shows that for each triple $v\,p\,w$ in H, H contains the triple p $\texttt{type}$ $\texttt{Property} = sur(I_A(p))$ $\texttt{type}$ $\texttt{Property}$, so that

$I_A(p) \in P_I$. In order to prove that I_A satisfies $v\,p\,w$, i.e. that $(I_A(v), I_A(w)) \in E_I(I_A(p))$, it is sufficient to prove that $v\ p\ sur(I_A(w)) \in H$, in view of Equation (28). Note that for each $l \in T(H) \cap L$, we have $sur(I_A(l)) = sur(I(l)) = b_l$. Combining this with Equation (28), it follows that if $v\,p\,w \in H$, then $v\ p\ sur(I_A(w)) \in H$, as desired: the only case that needs checking is where $w \in L$. In this case, rule lg shows that $v\ p\ b_w \in H$, as required: there is no entailment rule that can be applied with a literal in object position but that cannot be applied with a blank node in object position, so rule applications used to derive $v\,p\,w$ can be followed in parallel with other rule applications to provide a derivation of $v\ p\ b_w$.

Note that $I = S_K(G)$ satisfies all axiomatic triples, even though H may not contain all of them: if $i \notin K$ then $sur(\texttt{rdf:\_}i) = \texttt{rdf:\_}n$, so that the axiomatic triples including $\texttt{rdf:\_}i$ are also satisfied. For example, the triple $\texttt{rdf:\_}i$ `type` `Property` is satisfied because $sur(I(\texttt{rdf:\_}i))$ `type` `Property` $= \texttt{rdf:\_}n$ `type` `Property` $\in H$.

To prove the next condition on rdfs-interpretations, note that $v \in P_I$ if and only if $sur(v)$ `type` `Property` $\in H$ if and only if $(v, \texttt{Property}) \in E_I(\texttt{type})$.

We show that $a \in R_I$ if and only if $(a, \texttt{Resource}) \in E_I(\texttt{type})$. If $(a, I(\texttt{Resource})) \in E_I(\texttt{type})$, then clearly $a \in R_I$. Suppose that $a \in R_I$. It is sufficient to prove that $sur(a)$ `type` `Resource` $\in H$. If $a = val_D(l)$, $l \in V(G_s) \cap L_D^+$, then $sur(a) = b_l$ and H contains triples $v\,p\,l$ and $v\ p\ b_l$. Rule rdfs4b shows that $sur(a)$ `type` `Resource` $= b_l$ `type` `Resource` $\in H$. If $a = l \in V(G_s) \cap (L - L_D^+)$, then $sur(l) = b_l$ and H contains triples $v\,p\,l$ and $v\ p\ b_l$. Again, rule rdfs4b shows that $sur(l)$ `type` `Resource` $= b_l$ `type` `Resource` $\in H$. If

$$a \in V(G_s) \cap (U - (dom(D) \cup \{\texttt{rdf:\_}i : i \notin K\}))$$

or if $a \in bl(G_s)$, then $sur(a) = a$ and H contains, by rule rdf1, a triple of the form $a\,p\,v$ or $v\,p\,a$, so that by rule rdfs4a or rdfs4b, we have $sur(a)$ `type` `Resource` $= a$ `type` `Resource` $\in H$. If $a = \texttt{rdf:\_}i$ and $i \notin K$, then $sur(\texttt{rdf:\_}i)$ `type` `Resource` $= \texttt{rdf:\_}n$ `type` `Resource` $\in H$ in view of $\texttt{rdf:\_}n$ `type` `Property` $\in H$ and rule rdfs4a. If $a = d \in ran(D)$, then D contains a pair (a', d) and H contains the axiomatic triple a' `type` `Datatype`. Rule rdfs4a shows that H contains the triple $sur(d)$ `type` `Resource` $= a'$ `type` `Resource`.

To prove the next condition, note that $a \in LV_I$ if and only if $sur(a)$ `type` `Literal` $\in H$ if and only if $(a, \texttt{Literal}) \in E_I(\texttt{type})$.

Suppose next that $(a, b) \in E_I(\texttt{domain})$, $a \in P_I$, $b \in C_I$ and $(e, f) \in E_I(a)$. We need to show that $e \in CE_I(b)$, i.e. $(e, b) \in E_I(\texttt{type})$. We have $sur(a)$ `domain` $sur(b) \in H$. Since $(e, f) \in E_I(a)$, we conclude that $sur(e)\ sur(a)\ sur(f) \in H$. Rule rdfs2 shows that $sur(e)$ `type` $sur(b) \in H$, so that $(e, b) \in E_I(\texttt{type})$. The condition on $I(\texttt{range}) = \texttt{range}$ is proved in the same way, by means of rule

rdfs3.

We now prove that $E_I(\texttt{subPropertyOf})$ is transitive and reflexive on P_I. For $v \in P_I$, i.e. $sur(v)$ `type Property` $\in H$, we need to prove that $(v, v) \in E_I(\texttt{subPropertyOf})$, i.e. that $sur(v)$ `subPropertyOf` $sur(v) \in H$. This follows by rule rdfs6. If $(v, w) \in E_I(\texttt{subPropertyOf})$ and $(w, u) \in E_I(\texttt{subPropertyOf})$, then it follows that H contains the triples $sur(v)$ `subPropertyOf` $sur(w)$ and $sur(w)$ `subPropertyOf` $sur(u)$, so that by rule rdfs5, $sur(v)$ `subPropertyOf` $sur(u) \in H$ and $(v, u) \in E_I(\texttt{subPropertyOf})$.

We prove that if $(a, b) \in E_I(\texttt{subPropertyOf})$, then $a, b \in P_I$ and $E_I(a) \subseteq E_I(b)$. Note first that H contains the triple $sur(a)$ `subPropertyOf` $sur(b)$ as well as the axiomatic triples `subPropertyOf domain Property` and `subPropertyOf range Property`. Rules rdfs2 and rdfs3 show that the triples $sur(a)$ `type Property` and $sur(b)$ `type Property` are in H, so that $a \in P_I$ and $b \in P_I$. Suppose that $(e, f) \in E_I(a)$. Then we have $sur(e)$ $sur(a)$ $sur(f)$ $\in H$. Rule rdfs7x shows that $sur(e)$ $sur(b)$ $sur(f) \in H$ so that $(e, f) \in E_I(b)$. Note that the possibility is used that rule rdfs7x may produce a generalized RDF triple with a blank node in predicate position when applied to ordinary RDF triples.

The next condition to be proved is that if $a \in C_I$, then $(a, \texttt{Resource}) \in E_I(\texttt{subClassOf})$. Since $(a, \texttt{Class}) \in E_I(\texttt{type})$, we get $sur(a)$ `type Class` $\in H$. Rule rdfs8 shows that $sur(a)$ `subClassOf Resource` $\in H$, as required.

To prove that if $(a, b) \in E_I(\texttt{subClassOf})$ then $a, b \in C_I$ and $CE_I(a) \subseteq CE_I(b)$, note that H contains the triple $sur(a)$ `subClassOf` $sur(b)$ as well as the axiomatic triples `subClassOf domain Class` and `subClassOf range Class`. Rules rdfs2 and rdfs3 show that the triples $sur(a)$ `type Class` and $sur(b)$ `type Class` are in H, so that $(a, \texttt{Class})$, $(b, \texttt{Class}) \in E_I(\texttt{type})$ and $a, b \in C_I$. Suppose that $e \in CE_I(a)$. Then we have $(e, a) \in E_I(\texttt{type})$, so that $sur(e)$ `type` $sur(a) \in H$. Rule rdfs9 shows that $sur(e)$ `type` $sur(b) \in H$, so that $(e, b) \in E_I(\texttt{type})$ and $e \in CE_I(b)$.

We prove that $E_I(s_C)$ is transitive and reflexive on C_I. For $v \in C_I$, i.e. $(v, \texttt{Class}) \in E_I(\texttt{type})$, we need to prove that $(v, v) \in E_I(\texttt{subClassOf})$, i.e. that $sur(v)$ `subClassOf` $sur(v) \in H$. Since $sur(v)$ `type Class` $\in H$, rule rdfs10 shows that $sur(v)$ `subClassOf` $sur(v) \in H$. If $(v, w) \in E_I(\texttt{subClassOf})$ and $(w, u) \in E_I(\texttt{subClassOf})$, it follows that $sur(v)$ `subClassOf` $sur(w) \in H$ and $sur(w)$ `subClassOf` $sur(u) \in H$, so that by rule rdfs11, $sur(v)$ `subClassOf` $sur(u) \in H$ and $(v, u) \in E_I(\texttt{subClassOf})$.

In order to prove that if $a \in CE_I(\texttt{ContainerMembershipProperty})$, then $(a, \texttt{member}) \in E_I(\texttt{subPropertyOf})$, note that H contains the triple $sur(a)$ `type ContainerMembershipProperty`. Rule rdfs12 shows that H also contains the triple $sur(a)$ `subPropertyOf member`.

To prove the condition that if $a \in CE_I(\texttt{Datatype})$ then $(a, \texttt{Literal}) \in E_I(\texttt{subClassOf})$, note that H contains the triple $sur(a)$ `type Datatype`, so that rule rdfs13 shows that H also contains the triple $sur(a)$ `subClassOf Literal`.

We turn to the assumptions on D* interpretations and thereby also the assumptions on XML literals. Suppose that $(a, d) \in D$. Suppose that $l = (s, a) \in L_{\mathrm{t}} \cap V(G_s)$ and $s \in L(d)$. Then $L_I(l) = val_D(l) = L2V(d)(s) = A(b_l)$. In order to prove that $L_I(l) \in LV_I$, we need to prove that $sur(L_I(l))$ `type Literal` $= b_l$ `type Literal` $\in H$. Rule rdf2-D shows that the triple b_l `type` a is in H. With the axiomatic triple a `subClassOf Literal` and rule rdfs9 it follows that b_l `type Literal` $\in H$. We also have $(L_I(l), d) = (I_A(b_l), I_A(a)) \in E_I(I(\texttt{type}))$, since $sur(I_A(b_l))$ `type` $sur(I_A(a)) = b_l$ `type` $a \in H$.

Suppose that $(a, d) \in D$ and that $l = (s, a) \in V(G_{\mathrm{s}})$ is a D-literal that is not well typed. Then $A(b_l) = l$. Suppose that $(L_I(l), d) \in E_I(I(\texttt{type}))$; then b_l `type` $a = sur(L_I(l))$ `type` $sur(d) \in H$, so that with the axiomatic triple a `subClassOf Literal` and rule rdfs9 it follows that b_l `type Literal` $\in H$: H contains a D-clash. Suppose that $L_I(l) \in LV_I$; then, again, b_l `type Literal` $= sur(L_I(l))$ `type Literal` $\in H$: H contains a D-clash. □

4.11 D* entailment lemma. *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Then, $S \models_{\mathrm{s}} G$ if and only if there is a generalized RDF graph H that can be derived from $M(S)$ combined with RDF and RDFS axiomatic triples and D-axiomatic triples, by application of the D* entailment rules, and that either contains an instance of G as a subset or contains a D-clash.*

The preceding statement forms a direct extension, to D* entailment and generalized RDF graphs, of the RDFS entailment lemma [8]. The following statement gives a more precise version of the D* entailment lemma used to obtain decidability.

4.12 D* entailment lemma (alternative statement). *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Let H be a partial D* closure $M(S)_{\mathrm{s},K}$ of $M(S)$ and suppose that $i \in K$ for each* `rdf:_`$i \in V(G)$. *Then, $S \models_{\mathrm{s}} G$ if and only if either H contains an instance of G as a subset or H contains a D-clash.*

Proof of D entailment lemma (both statements).* $\Rightarrow$ Let $H = M(S)_{\mathrm{s},K}$ be a partial D* closure of $M(S)$ such that $i \in K$ for each `rdf:_`$i \in V(G)$, and let I be the associated D* Herbrand interpretation $S_K(M(S))$. (It is assumed that the set B of blank nodes is large enough to form the merge $M(S)$ and the partial closure H.) If H does not contain a D-clash, then I is a D* interpretation which satisfies H, so I satisfies S. By $S \models_{\mathrm{s}} G$, I satisfies G. This shows, in particular, that $V(G) \subseteq V(H)$. Choose $A : bl(G) \to R_I$ such that

I_A satisfies G. It follows that for each triple $s\,p\,o \in G$, $E_I(I_A(p))$ contains the pair $(I_A(s), I_A(o))$. Therefore, for each triple $s\,p\,o \in G$, H contains the triple

$$sur(I_A(s))\ sur(I_A(p))\ sur(I_A(o)).$$

The assumption that $i \in K$ for each $\texttt{rdf:\_}i \in V(G)$ implies that if $v \in V(G) \cap U$, then $sur(I(v)) = v$. Moreover, if $l \in V(G) \cap L$ then $sur(I(l)) = b_l$. Define the function $h : bl(G) \to T(H)$ to be $h = sur \circ A$, and extend h to all of $T(G)$ by $h(v) = v$ for each $v \in T(G) - bl(G)$. It follows that

- if $s\,p\,o \in G$ and o is not in L, then $h(s)\ h(p)\ h(o) \in H$
- if $l \in L$ and $s\,p\,l \in G$, then $h(s)\ h(p)\ b_l \in H$.

In the second of these two cases it also follows that $h(s)\ h(p)\ h(l) = h(s)\ h(p)\ l \in H$, in view of $V(G) \subseteq V(H)$ and rule gl. Hence, there is a subset of H that is an instance of G, with respect to the instance mapping h.

$\Leftarrow$ Suppose that the generalized RDF graph H can be derived from $M(S)$ combined with RDF and RDFS and D-axiomatic triples by application of the D* entailment rules. This holds, in particular, for any partial D* closure of $M(S)$. Suppose in addition that H either contains an instance of G as a subset or contains a D-clash. We prove that $S \models_{\mathrm{s}} G$. Let I be a D* interpretation that satisfies $M(S)$. By definition, I also satisfies all RDF and RDFS axiomatic triples and D-axiomatic triples. Choose the function $A' : bl(M(S)) \to R_I$ such that $I_{A'}$ satisfies $M(S)$. Define the function $A : bl(H) \to R_I$ by $A(b) = I(l)$ if b is a blank node allocated to the literal l by rule lg, and by $A(b) = A'(b)$ otherwise. We first show that we can prove that $S \models_{\mathrm{s}} G$ by showing that I_A satisfies H. If H contains an instance of G as a subset, then I satisfies G by the generalized instance lemma (Lemma 2.9), so that $S \models_{\mathrm{s}} G$. On the other hand, if H contains a D-clash, then S cannot have a satisfying D* interpretation, so that, again, $S \models_{\mathrm{s}} G$. To prove this, suppose that I is a D* interpretation that satisfies S. Then I_A as defined above satisfies a D-clash b `type` `Literal`, where the blank node b is allocated by rule lg to an ill-typed literal $l = (s, a) \in L_D - L_D^+$ of type $a \in dom(D)$. Therefore, $(L_I(l), I(\texttt{Literal})) = (A(b), I(\texttt{Literal})) \in E_I(I(\texttt{type}))$, so that $L_I(l) \in CE_I(I(\texttt{Literal})) = LV_I$. This contradicts the assumption that I is a D* interpretation.

We prove that I_A satisfies H by induction on the length of the derivations of the triples in H (see Definition 4.7). It is clear that I_A satisfies $M(S)$ and the RDF and RDFS and D-axiomatic triples. We next consider all D* entailment rules in turn.

Suppose that the triple $v\,p\,b$ is obtained from $v\,p\,l$ by means of rule lg, where b is allocated to l by rule lg. Then, $(I_A(v), I_A(b)) = (I_A(v), I_A(l)) \in E_I(I_A(p))$, so that I_A satisfies $v\,p\,b$.

Rule gl derives the triple $v\,p\,l$ from $v\,p\,b$, where $l \in L$ and where the blank node b is allocated to l by rule lg. Since I_A satisfies $v\,p\,b$, we get $(I_A(v), I_A(l)) = (I_A(v), I_A(b)) \in E_I(I_A(p))$.

Suppose that the triple p `type Property` is obtained from $v\,p\,w$ by means of rule rdf1. Then $I_A(p) \in P_I$ and, by a condition on rdfs-interpretations, $(I_A(p), I(\texttt{Property})) \in E_I(I_A(\texttt{type}))$, so that I satisfies p `type Property`.

Suppose that the triple b `type` a is obtained by an application of rule rdf2-D, given a triple $v\,p\,l$, where $l = (s, a)$ is a well-typed D-literal and where b is allocated to l by rule lg. Then $l \in V(S)$ and $(I_A(b), I_A(a)) = (L_I(l), I(a)) \in E_I(I(\texttt{type}))$ by a condition on D* interpretations, so that I_A satisfies b `type` a.

Assume that the triple b `type Literal` is obtained from the triple $v\,p\,l$ by rule rdfs1, where l is a plain literal and where the blank node b is allocated to l by rule lg. It follows that $(I_A(b), I(\texttt{Literal})) = (I(l), I(\texttt{Literal})) = (l, I(\texttt{Literal})) \in E_I(I(\texttt{type}))$, where in the last step we use $l \in LV_I = CE_I(I(\texttt{Literal}))$. Hence I_A satisfies b `type Literal`.

Turning to rule rdfs2, suppose that the triple v `type` u is obtained from p `domain` u and $v\,p\,w$. Then $(I_A(p), I_A(u)) \in E_I(I(\texttt{domain}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$, so that by Lemma 3.2, $I_A(v) \in CE_I(I_A(u))$, or $(I_A(v), I_A(u)) \in E_I(I(\texttt{type}))$, as required.

In order to prove the validity of rule rdfs3, suppose that the triple w `type` u is obtained from p `range` u and $v\,p\,w$, where $w \in U \cup B$. Then $(I_A(p), I_A(u)) \in E_I(I(\texttt{range}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$, so that by Lemma 3.2, $I_A(w) \in CE_I(I_A(u))$, or $(I_A(w), I_A(u)) \in E_I(I(\texttt{type}))$, as required.

For rule rdfs4a, we need to show that I_A satisfies v `type Resource` if it satisfies $v\,p\,w$. Since $I_A(v) \in R_I$ and $R_I = CE_I(I(\texttt{Resource}))$, it follows that $(I_A(v), I(\texttt{Resource})) \in E_I(I(\texttt{type}))$. For rule rdfs4b, we need to show that I_A satisfies w `type Resource` if it satisfies $v\,p\,w$ and if $w \in U \cup B$. Again, since $I_A(w) \in R_I$ and $R_I = CE_I(I(\texttt{Resource}))$, it follows that $(I_A(w), I(\texttt{Resource})) \in E_I(I(\texttt{type}))$.

For rule rdfs5, we need to show that I_A satisfies v `subPropertyOf` u if it satisfies v `subPropertyOf` w and w `subPropertyOf` u. We get $(I_A(v), I_A(w)) \in E_I(I(\texttt{subPropertyOf}))$ and $(I_A(w), I_A(u)) \in E_I(I(\texttt{subPropertyOf}))$, so that by a semantic condition on `subPropertyOf` we get $I_A(v), I_A(w), I_A(u) \in P_I$. Since $E_I(I(\texttt{subPropertyOf}))$ is transitive on P_I, it follows that $(I_A(v), I_A(u)) \in E_I(I(\texttt{subPropertyOf}))$.

Rule rdfs6 derives the triple v `subPropertyOf` v from v `type Property`. Since I_A satisfies v `type Property`, we have $(I_A(v), I(\texttt{Property})) \in E_I(I(\texttt{type}))$.

Hence $I_A(v) \in P_I$. Since $E_I(I(\texttt{subPropertyOf}))$ is reflexive on P_I, it follows that $(I_A(v), I_A(v)) \in E_I(I(\texttt{subPropertyOf}))$.

Rule rdfs7x derives the triple $v\,q\,w$ from p `subPropertyOf` q and $v\,p\,w$ if $q \in U \cup B$. We have $(I_A(p), I_A(q)) \in E_I(I(\texttt{subPropertyOf}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$. The second semantic condition on `subPropertyOf` shows that $(I_A(v), I_A(w)) \in E_I(I_A(q))$.

Rule rdfs8 derives the triple v `subClassOf Resource` from v `type Class`. Since $(I_A(v), I(\texttt{Class})) \in E_I(I(\texttt{type}))$, it follows that $I_A(v) \in CE_I(I(\texttt{Class})) = C_I$, so that $(I_A(v), I(\texttt{Resource})) \in E_I(I(\texttt{subClassOf}))$.

Rule rdfs9 derives the triple u `type` w from v `subClassOf` w and u `type` v. We have $(I_A(v), I_A(w)) \in E_I(I(\texttt{subClassOf}))$, so that $I_A(v) \in C_I$, $I_A(w) \in C_I$ and $CE_I(I_A(v)) \subseteq CE_I(I_A(w))$. Moreover, we have $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$, so that $I_A(u) \in CE_I(I_A(v))$ and hence $I_A(u) \in CE_I(I_A(w))$ and $(I_A(u), I_A(w)) \in E_I(I(\texttt{type}))$.

Rule rdfs10 is handled in the same way as rule rdfs6. Rule rdfs11 is handled in the same way as rule rdfs5.

Rule rdfs12 derives the triple v `subPropertyOf member` from the triple v `type ContainerMembershipProperty`. It is clear that $I_A(v) \in CE_I(I(\texttt{ContainerMembershipProperty}))$, so that the semantic condition on `ContainerMembershipProperty` shows that $(I_A(v), I(\texttt{member})) \in E_I(I(\texttt{subPropertyOf}))$.

Finally, rule rdfs13 derives the triple v `subClassOf Literal` from v `type Datatype`. Since $(I_A(v), I_A(\texttt{Datatype})) \in E_I(I(\texttt{type}))$, we get $I_A(v) \in CE_I(I(\texttt{Datatype}))$. Hence $(I_A(v), I(\texttt{Literal})) \in E_I(I(\texttt{subClassOf}))$. □

4.13 Corollary. *Let D be a finite datatype map. The D* entailment relation $S \models_{\mathrm{s}} G$ between finite sets S of finite generalized RDF graphs and finite generalized RDF graphs G is decidable. This problem is in NP, and in P if G is ground.*

Proof. In view of Lemma 4.12, the problem $S \models_{\mathrm{s}} G$ can be decided by taking a partial D* closure $H = M(S)_{\mathrm{s},K}$ of $M(S)$ such that K is finite and $i \in K$ for each `rdf:_`$i \in V(G)$, and by checking whether H contains a D-clash or whether H contains an instance of G as a subset. By Lemma 4.8, H can be computed in time polynomial in $|M(S)| + |G|$. The last statement now follows as in the case of simple entailment (see the remarks following the proof of Lemma 2.12). □

We shall now complement the above proof of decidability and NP-membership with a proof of NP-completeness of D* entailment (and RDFS entailment).

4.14 Proposition. *Let D be a finite datatype map. The D* entailment relation $S \models_{\mathrm{s}} G$ between finite sets S of finite generalized RDF graphs and finite generalized RDF graphs G is NP-complete.*

Proof. Membership of NP has already been shown above. As in the proof of Proposition 2.13, we prove NP-completeness using a reduction from the clique problem. An instance $G = (V, E), k$ of the clique problem (see the proof of Proposition 2.13) is transformed to RDF graphs G' and H'. The RDF graph H' consists, again, of the triples $v\,p\,w$ where p is a fixed property and where v and w are distinct elements of an arbitrary set of exactly k blank nodes. It is now also assumed that the URI reference p is not in the RDF and RDFS vocabulary or in the D-vocabulary. The RDF graph G' is formed in two steps. In the first step, each pair $\{v, w\} \in E$ is converted into two triples $v\,p\,w$ and $w\,p\,v$, where v and w are viewed as blank nodes. This leads to an RDF graph G''. In the second step, the graph G' is taken to be a partial D* closure of G''. In view of Lemma 4.8, is clear that the transformation from $G = (V, E), k$ to G', H' can be done in polynomial time. We need to prove that G has a clique of size $\geq k$ if and only if $G' \models_{\mathrm{s}} H'$. It follows in the same way as in the proof of Proposition 2.13 that G has a clique of size $\geq k$ if and only if there is a function $h : bl(H') \to bl(G'')$ satisfying $H'_h \subseteq G''$. We still have to prove that there is a function $h : bl(H') \to bl(G'')$ satisfying $H'_h \subseteq G''$ if and only if $G' \models_{\mathrm{s}} H'$. If there is a function $h : bl(H') \to bl(G'')$ satisfying $H'_h \subseteq G''$, then the interpolation lemma (Lemma 2.8) shows that $G'' \models H'$, so that by $G'' \subseteq G'$ we get $G' \models_{\mathrm{s}} H'$. To prove the converse implication, suppose that $G' \models_{\mathrm{s}} H'$. It is clear that G' itself is a partial D* closure of G', so the D* entailment lemma (Lemma 4.12) shows that there is a function $h : bl(H') \to nd(G')$ satisfying $H'_h \subseteq G'$. The proof can be concluded by showing that none of the triples in $G' - G''$ has p as predicate, for then it follows that $H'_h \subseteq G''$. The graph G'' has a simple structure: each of the triples in G'' has the same predicate p, which is not in the RDF and RDFS vocabulary or in the D-vocabulary, and all nodes in G'' are blank nodes. By considering the axiomatic triples and entailment rules it is not difficult to see that G' contains only three further triples containing p: rules rdf1, rdfs4a and rdfs6 produce the triples p `type` `Property`, p `type` `Resource` and p `subPropertyOf` p, respectively. (Similarly, each blank node x in G'' appears in exactly one new triple in G': rule rdfs4a or rdfs4b generates the triple x `type` `Resource`.) Therefore, none of the triples in $G' - G''$ has p as predicate. □

We conclude this section with a proposition summarizing a point that has been left implicit so far. A set S of generalized RDF graphs is called *D* consistent* if there is a D* interpretation that satisfies S.

4.15 Proposition. *Let D be a datatype map, S a set of generalized RDF graphs and H a partial D* closure of $M(S)$. Then, S is D* consistent if and only if H does not contain a D-clash. The problem to determine whether a*

finite set of finite generalized RDF graphs is D consistent is in P.*

Proof. The first of these two statements follows by Lemma 4.10 and the proof of Lemma 4.12 ($\Leftarrow$). The second statement follows from the first statement and Lemma 4.8. □

5 pD* Interpretations and pD* Entailment

In this section, we extend the above results on RDFS entailment and D* entailment to apply to a subset of the OWL vocabulary (see Table 5). We do not obtain the full power of the 'iff-semantics' (cf. 1.2) of OWL DL and OWL Full [14]. We consider a weaker semantics, called the pD* semantics, which extends RDFS and which is defined in a way analogous to the 'if-semantics' of RDFS. This leads to simple and useful entailment rules, which can be used to extend RDF reasoners. This weaker semantics, which is computationally less complex, seems to be sufficient for many applications of this subset of the OWL vocabulary (cf. 1.8). In the treatment of pD* entailment we follow the same pattern as in the preceding section for D* entailment. The definitions given in this section are refinements of the definitions given in the preceding section, incorporating semantic conditions for the subset of the OWL vocabulary considered.

5.1 The pD* semantics and OWL. Before turning to the formal definition of pD* interpretations, we explain the differences between the pD* semantics and the semantics of OWL (DL and Full) [14]. For `rdfs:subClassOf`, `rdfs:subPropertyOf`, `rdfs:domain` and `rdfs:range`, the pD* semantics adopts the if conditions of the RDFS semantics, rather than OWL's iff conditions for these constructs (see 1.2 for `rdfs:subClassOf`). Just like RDFS, the pD* semantics adds to these if conditions the conditions that `rdfs:subClassOf` and `rdfs:subPropertyOf` are reflexive and transitive; the latter conditions are implied by OWL's iff conditions. For `owl:FunctionalProperty`, `owl:InverseFunctionalProperty`, `owl:SymmetricProperty` and `owl:TransitiveProperty`, the well-known corresponding standard mathematical definitions (characterizing when binary relations are functional, inverse-functional, etc.) are used in if conditions, whereas OWL uses these standard mathematical definitions to form iff conditions. In particular, the pD* semantics requires that if a property is functional, then two values of the property for the same subject are always `owl:sameAs`-related, while OWL also requires the opposite condition. (As we shall see, unlike OWL, the pD* semantics does not always interpret `owl:sameAs` as equality.) The pD* semantics requires that if two classes are `owl:disjointWith`-related, then their extensions are disjoint, while the OWL semantics also requires the opposite condition. Moreover, the pD* semantics requires that if two properties are

Table 5
OWL URIs used in the pD* semantics.

owl:FunctionalProperty	owl:Restriction
owl:InverseFunctionalProperty	owl:onProperty
owl:SymmetricProperty	owl:hasValue
owl:TransitiveProperty	owl:someValuesFrom
owl:sameAs	owl:allValuesFrom
owl:inverseOf	owl:differentFrom
owl:equivalentClass	owl:disjointWith
owl:equivalentProperty	

owl:inverseOf-related, then their extensions are each other's inverse as binary relations, while OWL also requires the opposite condition. The pD* semantics requires that two classes are equivalent if and only if they are both subclasses of each other, while OWL requires that two classes are equivalent if and only if their extensions are equal; in combination with the pD* semantics for rdfs:subClassOf, it is not difficult to see that the pD* condition for owl:equivalentClass is somewhat stronger than the only-if part of OWL's condition for owl:equivalentClass. The pD* semantics treats owl:equivalentProperty in a similar way to owl:equivalentClass.

While OWL DL and OWL Full always interpret owl:sameAs as equality, the pD* semantics interprets owl:sameAs in a weaker way, as an equivalence relation. Several essential aspects of OWL's interpretation of sameAs are also obtained under the pD* semantics by adding further conditions. In particular, the pD* semantics requires that if two classes or properties are sameAs-related to each other, then these classes or properties have the same extension. Moreover, if the extension of a certain property includes a certain pair of resources, then this extension also contains any pair of resources obtained from the first pair of resources by substituting either of its two entries by another resource to which it is sameAs-related. The pD* semantics requires that the extensions of owl:sameAs and owl:differentFrom are disjoint, while OWL imposes the stronger condition that the extension of owl:differentFrom is the not-equal relation.

The if conditions of the pD* semantics share a common pattern, which can be described as follows: if certain instances satisfy certain conditions involving certain classes and/or properties, and if there is information about these classes and/or properties phrased in terms of the RDFS and/or OWL vocabulary, then these instances are required to satisfy certain other conditions. For the if conditions of the pD* semantics just discussed, OWL also imposes the opposite conditions, thus enabling conclusions to be drawn about entire classes and properties. With respect to the subset of the OWL vocabulary considered, the pD* semantics is intended to represent a reasonable interpretation that is useful for drawing conclusions about instances in the presence of an ontology (compare the examples discussed in 1.8), and that leads to simple entailment

rules and a relatively low computational complexity.

For `owl:hasValue`, `owl:someValuesFrom` and `owl:allValuesFrom`, the OWL semantics imposes certain if conditions which state equality of two sets in their then parts. These conditions can also be written as iff conditions. For example, OWL requires that a given resource belongs to an `hasValue` restriction with respect to a property and another resource if and only if this property has this other resource as value for the given resource. The pD* semantics includes this iff condition for `hasValue`. For `someValuesFrom` and `allValuesFrom`, the pD* semantics includes half of OWL's iff conditions. The condition of the pD* semantics for `allValuesFrom` requires that if a resource belongs to an `allValuesFrom` restriction with respect to a certain class and a certain property, then each value of this property for this resource belongs to this class; OWL also requires the opposite condition. Note that the direction of the if condition of the pD* semantics for `allValuesFrom` is similar to the direction of the if conditions of the pD* semantics for `subClassOf`, `FunctionalProperty`, etc.

The following if condition for `someValuesFrom` is part of OWL and is analogous to the if condition of the pD* semantics for `allValuesFrom` just mentioned: if a resource belongs to a `someValuesFrom` restriction with respect to a certain class and a certain property, then the resource is related with respect to this property to some other resource, while this other resource belongs to the given class. However, unlike all the if conditions of the pD* semantics (whose then parts make definite statements about instances already considered in the if parts), this condition for `someValuesFrom` has an existential quantification in its then part. Although this existential quantification can be handled with an entailment rule that introduces a new blank node, the proofs of Lemmas 4.8 and 5.8 do not extend to such rules. Therefore, we define a separate semantic extension of pD* entailment, called pD*sv entailment, which includes this condition for `someValuesFrom`. In Section 6 we show that the completeness result for pD* entailment obtained in this section can be extended to pD*sv entailment. The pD* semantics does include the other half of OWL's iff condition for `someValuesFrom`: for a `someValuesFrom` restriction with respect to a certain class and a certain property, and for a resource, the pD* semantics requires that if there is a value for this property belonging to the class, then this resource belongs to the `someValuesFrom` restriction. As has already been mentioned in 1.8, the pD* semantics for `someValuesFrom` may not be sufficient for various applications. See 1.8 for a discussion of an alternative, more controlled way of introducing new blank nodes in connection with `someValuesFrom`, which may be useful in some situations in combination with the use of a reasoner that supports only the pD* semantics.

5.2 Definition (pD* interpretations). The *P-vocabulary*, denoted by pV, is a set of URI references from the OWL vocabulary [14] listed in Table 5.

Let D be a datatype map. A *pD* interpretation* of a vocabulary V is a D* interpretation I of $V \cup pV$ that satisfies the following conditions:

- I satisfies the triples in Table 6, which are called the *P-axiomatic triples.*
- If $p \in CE_I(I(\texttt{FunctionalProperty}))$ and (a,b), $(a,c) \in E_I(p)$, then $(b,c) \in E_I(I(\texttt{sameAs}))$
- If $p \in CE_I(I(\texttt{InverseFunctionalProperty}))$ and (a,c), $(b,c) \in E_I(p)$, then $(a,b) \in E_I(I(\texttt{sameAs}))$
- If $p \in CE_I(I(\texttt{SymmetricProperty}))$ and $(a,b) \in E_I(p)$, then $(b,a) \in E_I(p)$
- If $p \in CE_I(I(\texttt{TransitiveProperty}))$ and (a,b), $(b,c) \in E_I(p)$, then $(a,c) \in E_I(p)$
- $E_I(I(\texttt{sameAs}))$ is an equivalence relation on R_I
- If $a \in C_I$ and $(a,b) \in E_I(I(\texttt{sameAs}))$, then $(a,b) \in E_I(I(\texttt{subClassOf}))$
- If $p \in P_I$ and $(p,q) \in E_I(I(\texttt{sameAs}))$, then $(p,q) \in E_I(I(\texttt{subPropertyOf}))$
- If $p \in P_I$, $(a,b) \in E_I(p)$, $(a,a') \in E_I(I(\texttt{sameAs}))$ and $(b,b') \in E_I(I(\texttt{sameAs}))$, then $(a',b') \in E_I(p)$
- If $(p,q) \in E_I(I(\texttt{inverseOf}))$ then $(a,b) \in E_I(p)$ if and only if $(b,a) \in E_I(q)$
- $(a,b) \in E_I(I(\texttt{equivalentClass}))$ if and only if $(a,b) \in E_I(I(\texttt{subClassOf}))$ and $(b,a) \in E_I(I(\texttt{subClassOf}))$
- $(a,b) \in E_I(I(\texttt{equivalentProperty}))$ if and only if $(a,b) \in E_I(I(\texttt{subPropertyOf}))$ and $(b,a) \in E_I(I(\texttt{subPropertyOf}))$
- If $(a,b) \in E_I(I(\texttt{hasValue}))$ and $(a,p) \in E_I(I(\texttt{onProperty}))$, then $x \in CE_I(a)$ if and only if $(x,b) \in E_I(p)$
- If $(a,b) \in E_I(I(\texttt{someValuesFrom}))$, $(a,p) \in E_I(I(\texttt{onProperty}))$, $(x,y) \in E_I(p)$ and $y \in CE_I(b)$, then $x \in CE_I(a)$
- If $(a,b) \in E_I(I(\texttt{allValuesFrom}))$, $(a,p) \in E_I(I(\texttt{onProperty}))$, $x \in CE_I(a)$ and $(x,y) \in E_I(p)$, then $y \in CE_I(b)$
- $E_I(I(\texttt{differentFrom})) \cap E_I(I(\texttt{sameAs})) = \emptyset$
- If $(a,b) \in E_I(I(\texttt{disjointWith}))$, then $CE_I(a) \cap CE_I(b) = \emptyset$

5.3 Lemma. *pD* interpretations are well defined, in the sense that each invocation of the functions E_I and CE_I in the definition is allowed in view of the earlier conditions. If I is a pD* interpretation of a vocabulary V, then*

- $I(\texttt{sameAs})$, $I(\texttt{inverseOf})$, $I(\texttt{equivalentClass})$, $I(\texttt{equivalentProperty})$, $I(\texttt{onProperty})$, $I(\texttt{hasValue})$, $I(\texttt{someValuesFrom})$, $I(\texttt{allValuesFrom})$, $I(\texttt{differentFrom})$ *and* $I(\texttt{disjointWith})$ *are in* P_I
- $I(\texttt{FunctionalProperty})$, $I(\texttt{InverseFunctionalProperty})$, $I(\texttt{SymmetricProperty})$, $I(\texttt{TransitiveProperty})$ *and* $I(\texttt{Restriction})$ *are in* C_I

Proof. Suppose that I is a D* interpretation of $V \cup pV$ that satisfies the P-axiomatic triples. Since I satisfies the triples `sameAs type Property`, `inverseOf type Property`, `equivalentClass type Property`, `equivalentProperty type Property` and `differentFrom type Property`, it follows that

Table 6
P-axiomatic triples.

`FunctionalProperty subClassOf Property .`
`InverseFunctionalProperty subClassOf Property .`
`SymmetricProperty subClassOf Property .`
`TransitiveProperty subClassOf Property .`
`sameAs type Property .`
`inverseOf type Property .`
`inverseOf domain Property .`
`inverseOf range Property .`
`equivalentClass type Property .`
`equivalentProperty type Property .`
`equivalentClass domain Class .`
`equivalentClass range Class .`
`equivalentProperty domain Property .`
`equivalentProperty range Property .`
`Restriction subClassOf Class .`
`onProperty domain Restriction .`
`onProperty range Property .`
`hasValue domain Restriction .`
`someValuesFrom domain Restriction .`
`someValuesFrom range Class .`
`allValuesFrom domain Restriction .`
`allValuesFrom range Class .`
`differentFrom type Property .`
`disjointWith domain Class .`
`disjointWith range Class .`

$I(\texttt{sameAs})$, $I(\texttt{inverseOf})$, $I(\texttt{equivalentClass})$, $I(\texttt{equivalentProperty})$, $I(\texttt{differentFrom})$ are all in $CE_I(I(\texttt{Property})) = P_I$.

Since I satisfies the triple `FunctionalProperty subClassOf Property`, a semantic condition on `subClassOf` shows that we have $I(\texttt{FunctionalProperty}) \in C_I$ and $CE_I(I(\texttt{FunctionalProperty})) \subseteq CE_I(I(\texttt{Property})) = P_I$. The cases of `InverseFunctionalProperty`, `SymmetricProperty` and `TransitiveProperty` are handled in the same way.

Since I satisfies `Restriction subClassOf Class`, we get $I(\texttt{Restriction}) \in C_I$. Since I satisfies `onProperty domain Restriction`, we have $(I(\texttt{onProperty}), I(\texttt{Restriction})) \in E_I(I(\texttt{domain}))$, so that by Lemma 3.2, $I(\texttt{onProperty}) \in P_I$. It follows in the same way that $I(\texttt{hasValue})$, $I(\texttt{someValuesFrom})$, $I(\texttt{allValuesFrom}) \in P_I$.

If $(p, q) \in E_I(I(\texttt{inverseOf}))$, then since I satisfies `inverseOf domain Property` and `inverseOf range Property`, it follows by Lemma 3.2 that $p, q \in CE_I(I(\texttt{Property})) = P_I$.

In order to show that if $(a, b) \in E_I(I(\texttt{hasValue}))$ then $a \in C_I$, note that the axiomatic triple `hasValue domain Restriction` shows that $(I(\texttt{hasValue}), I(\texttt{Restriction})) \in E_I(I(\texttt{domain}))$ so that again by Lemma 3.2, $a \in CE_I(I(\texttt{Restriction}))$. The axiomatic triple `Restriction subClassOf Class` shows that $(I(\texttt{Restriction}), I(\texttt{Class})) \in E_I(I(\texttt{subClassOf}))$, so that $a \in CE_I(I(\texttt{Class})) = C_I$. In a similar way, it follows that if $(a, b) \in E_I(I(\texttt{someValuesFrom}))$ or $(a, b) \in E_I(I(\texttt{allValuesFrom}))$, then $a \in C_I$.

In order to show that if $(a, p) \in E_I(I(\texttt{onProperty}))$ then $p \in P_I$, note that by the axiomatic triple `onProperty range Property` we have $(I(\texttt{onProperty}), I(\texttt{Property})) \in E_I(I(\texttt{range}))$, so that by Lemma 3.2, $p \in CE_I(I(\texttt{Property})) = P_I$.

In order to show that if $(a, b) \in E_I(I(\texttt{someValuesFrom}))$ then $b \in C_I$, note that the axiomatic triple `someValuesFrom range Class` shows that $(I(\texttt{someValuesFrom}), I(\texttt{Class})) \in E_I(I(\texttt{range}))$. It follows in the same way that if $(a, b) \in E_I(I(\texttt{allValuesFrom}))$, then $b \in C_I$.

Since I satisfies `disjointWith domain Class` and `disjointWith range Class`, it follows that $I(\texttt{disjointWith}) \in P_I$ and that if $(a, b) \in E_I(I(\texttt{disjointWith}))$ then $a, b \in C_I$. □

5.4 Definition (pD* entailment). Given a datatype map D, a set S of generalized RDF graphs *pD* entails* a generalized RDF graph G if each pD* interpretation I that satisfies S also satisfies G. In this case, we write

$$S \models_{\mathrm{p}} G.$$

It is clear that each OWL Full interpretation [14] is a pD* interpretation, so that if $S \models_{\mathrm{p}} G$, then S OWL Full entails G. Here we extend the definition of OWL Full entailment [8] to generalized RDF graphs in the following way.

5.5 Definition (OWL Full entailment). Given a datatype map D, a set S of generalized RDF graphs *OWL Full entails* a generalized RDF graph G if each OWL Full interpretation I that satisfies S also satisfies G.

5.6 Definition (P-entailment rules). See Table 7 for the definition of the P-entailment rules. Note that rules rdfp8ax, rdfp8bx and rdfp14bx are the only rules in this table that can produce generalized RDF triples with blank nodes in predicate position (see 2.2) when applied to ordinary RDF triples.

5.7 Definition (partial and full pD* closures). Suppose that G is an RDF graph and D a datatype map. The *partial pD* closure* $G_{\mathrm{p},K}$ of G, and the *full pD* closure* G_{p} of G, are defined in a way similar to the definition of the partial and full D* closures $G_{\mathrm{s},K}$ and G_{s} of G (see Definition 4.6). The only

Table 7
P-entailment rules.

	If G contains	where	then add to G
rdfp1	p `type` `FunctionalProperty` $u\ p\ v$ $u\ p\ w$	$v \in U \cup B$	v `sameAs` w
rdfp2	p `type` `Inverse-FunctionalProperty` $u\ p\ w$ $v\ p\ w$		u `sameAs` v
rdfp3	p `type` `SymmetricProperty` $v\ p\ w$	$w \in U \cup B$	$w\ p\ v$
rdfp4	p `type` `TransitiveProperty` $u\ p\ v$ $v\ p\ w$		$u\ p\ w$
rdfp5a	$v\ p\ w$		v `sameAs` v
rdfp5b	$v\ p\ w$	$w \in U \cup B$	w `sameAs` w
rdfp6	v `sameAs` w	$w \in U \cup B$	w `sameAs` v
rdfp7	u `sameAs` v v `sameAs` w		u `sameAs` w
rdfp8ax	p `inverseOf` q $v\ p\ w$	$w, q \in U \cup B$	$w\ q\ v$
rdfp8bx	p `inverseOf` q $v\ q\ w$	$w \in U \cup B$	$w\ p\ v$
rdfp9	v `type` `Class` v `sameAs` w		v `subClassOf` w
rdfp10	p `type` `Property` p `sameAs` q		p `subPropertyOf` q
rdfp11	$u\ p\ v$ u `sameAs` u' v `sameAs` v'	$u' \in U \cup B$	$u'\ p\ v'$
rdfp12a	v `equivalentClass` w		v `subClassOf` w
rdfp12b	v `equivalentClass` w	$w \in U \cup B$	w `subClassOf` v
rdfp12c	v `subClassOf` w w `subClassOf` v		v `equivalent-Class` w
rdfp13a	v `equivalentProperty` w		v `subPropertyOf` w
rdfp13b	v `equivalentProperty` w	$w \in U \cup B$	w `subPropertyOf` v
rdfp13c	v `subPropertyOf` w w `subPropertyOf` v		v `equivalent-Property` w
rdfp14a	v `hasValue` w v `onProperty` p $u\ p\ w$		u `type` v
rdfp14bx	v `hasValue` w v `onProperty` p u `type` v	$p \in U \cup B$	$u\ p\ w$
rdfp15	v `someValuesFrom` w v `onProperty` p $u\ p\ x$ x `type` w		u `type` v
rdfp16	v `allValuesFrom` w v `onProperty` p u `type` v $u\ p\ x$	$x \in U \cup B$	x `type` w

differences are in the first and last steps. In the first step, the P-axiomatic triples are also added to G. In the last step, the P-entailment rules are used as well. Just like a partial D* closure, a partial pD* closure is, in general, a generalized RDF graph, even if the given graph is an ordinary RDF graph.

5.8 Lemma. *Let D be a finite datatype map. If G is a finite generalized RDF graph, then each partial pD* closure $G_{\mathrm{p},K}$ of G is finite for K finite, and a partial pD* closure of G can be computed in polynomial time.*

Proof. This is proved as for partial D* closures (see Lemma 4.8), adding the P-axiomatic triples in the first step. For the last part of the proof, note that for rules rdfp1, rdfp2, rdfp4, rdfp12, rdfp14a and rdfp14bx the existence of a successful rule application can be detected in time $O(n^3)$, where n is the size of the partial closure graph under construction. Since it requires two triples to define a `someValuesFrom` or `allValuesFrom` restriction, the complexity is higher for rules rdfp15 and rdfp16: these rules can be handled in time $O(n^4)$. The other P-entailment rules can be handled in linear or quadratic time, just like the D* entailment rules. □

5.9 Definition (pD* Herbrand interpretation). Given a datatype map D and a generalized RDF graph G, a *pD* Herbrand interpretation* $P_K(G)$ is defined in a similar way to a D* Herbrand interpretation $S_K(G)$ (see Definition 4.9). The only differences are that G_{s} is replaced by G_{p} and $G_{\mathrm{s},K}$ is replaced by $G_{\mathrm{p},K}$.

5.10 Definition (P-clash). In addition to D-clashes (see Definition 4.5), the pD* semantics also leads to possible inconsistencies in connection with `differentFrom` and `disjointWith`: a *P-clash* is either a combination of two triples of the form v `differentFrom` w, v `sameAs` w, or a combination of three triples of the form v `disjointWith` w, u `type` v, u `type` w.

5.11 pD* satisfaction lemma. *Let G be an RDF graph and D a datatype map. If the partial pD* closure $G_{\mathrm{p},K}$ of G does not contain a P-clash or a D-clash, then $P_K(G)$ is a pD* interpretation that satisfies $G_{\mathrm{p},K}$.*

Proof. In this proof the pD* Herbrand interpretation $P_K(G)$ of G is denoted by I and the partial pD* closure $G_{\mathrm{p},K}$ of G by H. Several additions need to be made to the proof of the D* satisfaction lemma (see Lemma 4.10). It is clear that I satisfies the P-axiomatic triples.

We prove the condition that if $p \in CE_I(I(\texttt{FunctionalProperty}))$ and (a, b), $(a, c) \in E_I(p)$, then $(b, c) \in E_I(I(\texttt{sameAs}))$. We have $(p, \texttt{FunctionalProperty}) \in E_I(\texttt{type})$, so that H contains $sur(p)$ `type` `FunctionalProperty`. We also have $sur(a)$ $sur(p)$ $sur(b) \in H$ and $sur(a)$ $sur(p)$ $sur(c) \in H$, so that by rule rdfp1, $sur(b)$ `sameAs` $sur(c) \in H$, as desired. It can be proved in a similar way that if $p \in$

$CE_I(I(\texttt{InverseFunctionalProperty}))$ and (a,c), $(b,c) \in E_I(p)$, then $(a,b) \in E_I(I(\texttt{sameAs}))$.

In order to prove that if $p \in CE_I(I(\texttt{SymmetricProperty}))$ and $(a,b) \in E_I(p)$ then $(b,a) \in E_I(p)$, suppose that $(p, \texttt{SymmetricProperty}) \in E_I(\texttt{type})$, so that $sur(p)$ `type` `SymmetricProperty` $\in H$. We can also conclude that $sur(a)$ $sur(p)$ $sur(b) \in H$, so that by rule rdfp3, $sur(b)$ $sur(p)$ $sur(a) \in H$.

We prove that if $p \in CE_I(I(\texttt{TransitiveProperty}))$ and (a,b), $(b,c) \in E_I(p)$, then $(a,c) \in E_I(p)$. We have $(p, \texttt{TransitiveProperty}) \in E_I(\texttt{type})$, so $sur(p)$ `type` `TransitiveProperty` $\in H$. H also contains the triples $sur(a)$ $sur(p)$ $sur(b)$ and $sur(b)$ $sur(p)$ $sur(c)$. By rule rdfp4, $sur(a)$ $sur(p)$ $sur(c) \in H$, as desired.

To prove reflexivity of $E_I(I(\texttt{sameAs}))$ on R_I, the five parts of the definition of R_I (see Definition 4.9) need to be considered separately. Suppose first that $l \in V(G_\mathrm{p}) \cap L_D^+$. Then by rules lg and rdfp5b, H contains the triple b_l `sameAs` $b_l = sur(val_D(l))$ `sameAs` $sur(val_D(l))$, so that $(val_D(l), val_D(l)) \in E_I(\texttt{sameAs})$, as desired. Suppose next that $l \in V(G_\mathrm{p}) \cap (L - L_D^+)$. Again by rules lg and rdfp5b, H contains the triple b_l `sameAs` $b_l = sur(l)$ `sameAs` $sur(l)$, so that $(l,l) \in E_I(I(\texttt{sameAs}))$, as desired. Suppose next that

$$a \in V(G_\mathrm{p}) \cap (U - (dom(D) \cup \{\texttt{rdf:\_}i : i \notin K\}))$$

or $a \in bl(G_p)$. Then, $sur(a) = a$ and H contains, by rule rdf1, a triple of the form $a\,p\,v$ or $v\,p\,a$, so that by rules rdfp5a and rdfp5b it follows that H contains the triple a `sameAs` $a = sur(a)$ `sameAs` $sur(a)$, so that $(a,a) \in E_I(I(\texttt{sameAs}))$. If $i \notin K$, then $sur(\texttt{rdf:\_}i) = \texttt{rdf:\_}n$ and H contains the axiomatic triple $\texttt{rdf:\_}n$ `type` `Property`, so that by rule rdfp5a $sur(\texttt{rdf:\_}i)$ `sameAs` $sur(\texttt{rdf:\_}i) = \texttt{rdf:\_}n$ `sameAs` $\texttt{rdf:\_}n \in H$ and $(\texttt{rdf:\_}i, \texttt{rdf:\_}i) \in E_I(I(\texttt{sameAs}))$. If $d \in ran(D)$, then D contains a pair (a,d) and H contains the axiomatic triple a `type` `Datatype`. In view of rule rdfp5a, it follows that H contains the triple a `sameAs` $a = sur(d)$ `sameAs` $sur(d)$, so that $(d,d) \in E_I(I(\texttt{sameAs}))$. This concludes the proof of reflexivity.

To prove symmetry of $E_I(I(\texttt{sameAs}))$, suppose that $(a,b) \in E_I(I(\texttt{sameAs}))$. Then H contains $sur(a)$ `sameAs` $sur(b)$ and by rule rdfp6 also $sur(b)$ `sameAs` $sur(a)$, so that $(b,a) \in E_I(I(\texttt{sameAs}))$. To prove transitivity of $E_I(I(\texttt{sameAs}))$, suppose that (a,b), $(b,c) \in E_I(I(\texttt{sameAs}))$. Then H contains $sur(a)$ `sameAs` $sur(b)$ and $sur(b)$ `sameAs` $sur(c)$ and, by rule rdfp7, also $sur(a)$ `sameAs` $sur(c)$, so that $(a,c) \in E_I(I(\texttt{sameAs}))$.

Suppose that $a \in C_I$ and $(a,b) \in E_I(I(\texttt{sameAs}))$. In order to prove that $(a,b) \in E_I(I(\texttt{subClassOf}))$, note that by $C_I = CE_I(I(\texttt{Class}))$ we have $(a, \texttt{Class}) \in E_I(\texttt{type})$, so that $sur(a)$ `type` `Class` $\in H$. Moreover, $sur(a)$ `sameAs` $sur(b)$ $\in H$, so that by rule rdfp9, $sur(a)$ `subClassOf` $sur(b) \in H$, as desired. We

can prove in the same way, using rule rdfp10, that if $p \in P_I$ and $(p, q) \in E_I(I(\texttt{sameAs}))$ then $(p, q) \in E_I(I(\texttt{subPropertyOf}))$.

Suppose that $p \in P_I$, $(a, b) \in E_I(p)$, $(a, a') \in E_I(I(\texttt{sameAs}))$ and $(b, b') \in E_I(I(\texttt{sameAs}))$. In order to prove that $(a', b') \in E_I(p)$, note that H contains the triples $sur(a)$ $sur(p)$ $sur(b)$, $sur(a)$ `sameAs` $sur(a')$ and $sur(b)$ `sameAs` $sur(b')$. Rule rdfp11 shows that H also contains the triple $sur(a')$ $sur(p)$ $sur(b')$.

We prove that if $(p, q) \in E_I(I(\texttt{inverseOf}))$, then $(a, b) \in E_I(p)$ if and only if $(b, a) \in E_I(q)$. We have $sur(p)$ `inverseOf` $sur(q) \in H$. Therefore, by rules rdfp8ax and rdfp8bx, $sur(a)$ $sur(p)$ $sur(b) \in H$ if and only if $sur(b)$ $sur(q)$ $sur(a) \in H$.

We prove that if $(a, b) \in E_I(I(\texttt{equivalentClass}))$, then $(a, b) \in E_I(I(\texttt{subClassOf}))$ and $(b, a) \in E_I(I(\texttt{subClassOf}))$. By $sur(a)$ `equivalentClass` $sur(b) \in H$ and rules rdfp12a and rdfp12b, the triples $sur(a)$ `subClassOf` $sur(b)$ and $sur(b)$ `subClassOf` $sur(a)$ are in H, as required. To show the converse, suppose that H contains $sur(a)$ `subClassOf` $sur(b)$ and $sur(b)$ `subClassOf` $sur(a)$. Then rule rdfp12c shows that $sur(a)$ `equivalentClass` $sur(b) \in H$, so that $(a, b) \in E_I(I(\texttt{equivalentClass}))$. The iff condition connecting `equivalentProperty` and `subPropertyOf` is proved in the same way.

We prove that if $(p, q) \in E_I(I(\texttt{inverseOf}))$ then $(a, b) \in E_I(p)$ if and only if $(b, a) \in E_I(q)$. We have $sur(p)$ `inverseOf` $sur(q) \in H$. Therefore, by rules rdfp8ax and rdfp8bx, $sur(a)$ $sur(p)$ $sur(b) \in H$ if and only if $sur(b)$ $sur(q)$ $sur(a) \in H$.

Suppose that $(a, b) \in E_I(I(\texttt{hasValue}))$ and $(a, p) \in E_I(I(\texttt{onProperty}))$, so that the triples $sur(a)$ `hasValue` $sur(b)$ and $sur(a)$ `onProperty` $sur(p)$ are in H. To prove that $x \in CE_I(a)$ if and only if $(x, b) \in E_I(p)$, note that by rules rdfp14a and rdfp14bx, $sur(x)$ `type` $sur(a) \in H$ if and only if $sur(x)$ $sur(p)$ $sur(b) \in H$.

We prove that if $(a, b) \in E_I(I(\texttt{someValuesFrom}))$, $(a, p) \in E_I(I(\texttt{onProperty}))$, $(x, y) \in E_I(p)$ and $y \in CE_I(b)$, then $x \in CE_I(a)$. Note that H contains the triples $sur(a)$ `someValuesFrom` $sur(b)$, $sur(a)$ `onProperty` $sur(p)$, $sur(x)$ $sur(p)$ $sur(y)$ and $sur(y)$ `type` $sur(b)$. Rule rdfp15 shows that we have $sur(x)$ `type` $sur(a) \in H$, so that $(x, a) \in E_I(I(\texttt{type}))$, as desired.

We prove that if $(a, b) \in E_I(I(\texttt{allValuesFrom}))$, $(a, p) \in E_I(I(\texttt{onProperty}))$, $x \in CE_I(a)$ and $(x, y) \in E_I(p)$, then $y \in CE_I(b)$. Note that H contains the triples $sur(a)$ `allValuesFrom` $sur(b)$, $sur(a)$ `onProperty` $sur(p)$, $sur(x)$ `type` $sur(a)$ and $sur(x)$ $sur(p)$ $sur(y)$. Rule rdfp16 shows that we have $sur(y)$ `type` $sur(b) \in H$, so that $(y, b) \in E_I(I(\texttt{type}))$, as desired.

Suppose that I does not satisfy the condition on `differentFrom`. Then, there exists a pair $(a, b) \in E_I(I(\texttt{differentFrom})) \cap E_I(I(\texttt{sameAs}))$, showing that H contains the triples $sur(a)$ `differentFrom` $sur(b)$ and $sur(a)$ `sameAs` $sur(b)$. In other words, H contains a P-clash. Suppose that I does not satisfy the condition on `disjointWith`. It follows that there exists a pair $(a, b) \in E_I(I(\texttt{disjointWith}))$ and a resource $c \in CE_I(a) \cap CE_I(b)$, showing again that H contains a P-clash, which consists of the triples $sur(a)$ `disjointWith` $sur(b)$, $sur(c)$ `type` $sur(a)$ and $sur(c)$ `type` $sur(b)$. □

5.12 pD* entailment lemma. *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Then, $S \models_{\mathrm{p}} G$ if and only if there is a generalized RDF graph H that can be derived from $M(S)$ combined with RDF and RDFS axiomatic triples and D-axiomatic triples and P-axiomatic triples, by application of D* entailment rules and P-entailment rules, and that either contains an instance of G as a subset or contains a P-clash or a D-clash.*

5.13 pD* entailment lemma (alternative statement). *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Let H be a partial pD* closure $M(S)_{\mathrm{p},K}$ of $M(S)$ and suppose that $i \in K$ for each `rdf:_`$i \in V(G)$. Then, $S \models_{\mathrm{p}} G$ if and only if either H contains an instance of G as a subset or H contains a P-clash or a D-clash.*

Proof. The proof of the pD* entailment lemma builds further on the proof of the D* entailment lemma (Lemmas 4.11 and 4.12). For the only if direction, the pD* satisfaction lemma is used instead of the D* satisfaction lemma. For the if direction we use the interpretation I and the associated function I_A as described in the earlier proof. It will be shown that I_A satisfies H. This will again conclude the proof, because if H contains a P-clash, then I cannot be a pD* interpretation. To show this, suppose first that H contains the P-clash v `differentFrom` w, v `sameAs` w. Then $(I_A(v), I_A(w)) \in E_I(I(\texttt{differentFrom})) \cap E_I(I(\texttt{sameAs}))$, contradicting a requirement on pD* interpretations. Suppose next that H contains the P-clash v `disjointWith` w, u `type` v, u `type` w. Then $(I_A(v), I_A(w)) \in E_I(I(\texttt{disjointWith}))$ and $I_A(u) \in CE_I(I_A(v)) \cap CE_I(I_A(w))$, which again contradicts a requirement on pD* interpretations. It remains only to prove the validity of the P-entailment rules. We continue the earlier proof that I_A satisfies H by induction on the length of derivations of triples in H.

Suppose that rule rdfp1 is used to obtain the triple v `sameAs` w from p `type FunctionalProperty`, $u\,p\,v$ and $u\,p\,w$, where $v \in U \cup B$. Then we have $I_A(p) \in CE_I(I(\texttt{FunctionalProperty}))$, $(I_A(u), I_A(v)) \in E_I(I_A(p))$ and $(I_A(u), I_A(w)) \in E_I(I_A(p))$, so that $(I_A(v), I_A(w)) \in E_I(I(\texttt{sameAs}))$ and I_A satisfies v `sameAs` w. The condition on $I(\texttt{InverseFunctionalProperty})$ is used in a similar way to obtain the validity of rule rdfp2.

Assume that $w\,p\,v$ is obtained with rule rdfp3 from p `type SymmetricProperty` and $v\,p\,w$, where $w \in U \cup B$. Then we have $I_A(p) \in CE_I(I(\texttt{SymmetricProperty}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$, so that $(I_A(w), I_A(v)) \in E_I(I_A(p))$ and I_A satisfies $w\,p\,v$.

Suppose that $u\,p\,w$ is obtained from p `type TransitiveProperty`, $u\,p\,v$ and $v\,p\,w$ by means of rule rdfp4. Then we have $I_A(p) \in CE_I(I(\texttt{TransitiveProperty}))$ and $(I_A(u), I_A(v))$, $(I_A(v), I_A(w)) \in E_I(I_A(p))$, so that $(I_A(u), I_A(w)) \in E_I(I_A(p))$ and I_A satisfies $u\,p\,w$.

As to rule rdfp5a, note that if I_A satisfies $v\,p\,w$, then $(I_A(v), I_A(v)) \in E_I(I(\texttt{sameAs}))$, so that I_A satisfies v `sameAs` v. Similarly, for rule rdfp5b, if I_A satisfies $v\,p\,w$, where $w \in U \cup B$, then $(I_A(w), I_A(w)) \in E_I(I(\texttt{sameAs}))$, so that I_A satisfies w `sameAs` w.

Suppose that w `sameAs` v is obtained from v `sameAs` w with rule rdfp6, where $w \in U \cup B$. Then $(I_A(v), I_A(w)) \in E_I(I(\texttt{sameAs}))$, so that $(I_A(w), I_A(v)) \in E_I(I(\texttt{sameAs}))$ as required. Suppose that rule rdfp7 is used to derive u `sameAs` w from u `sameAs` v and v `sameAs` w. Then $(I_A(u), I_A(v))$, $(I_A(v), I_A(w)) \in E_I(I(\texttt{sameAs}))$ and therefore $(I_A(u), I_A(w)) \in E_I(I(\texttt{sameAs}))$, as required.

Rule rdfp8ax is used to obtain $w\,q\,v$ from p `inverseOf` q and $v\,p\,w$, where $w, q \in U \cup B$. We have $(I_A(p), I_A(q)) \in E_I(I(\texttt{inverseOf}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(p))$, so that $(I_A(w), I_A(v)) \in E_I(I_A(q))$. Suppose that $w\,p\,v$ is obtained from p `inverseOf` q and $v\,q\,w$ with rule rdfp8bx, where $w \in U \cup B$. Then $(I_A(p), I_A(q)) \in E_I(I(\texttt{inverseOf}))$ and $(I_A(v), I_A(w)) \in E_I(I_A(q))$, so that $(I_A(w), I_A(v)) \in E_I(I_A(p))$.

Suppose that rule rdfp9 is used to obtain v `subClassOf` w from v `type Class` and v `sameAs` w. Then, $(I_A(v), I(\texttt{Class})) \in E_I(I(\texttt{type}))$, so that by Lemma 3.2, $I_A(v) \in CE_I(I(\texttt{Class})) = C_I$. Moreover, it follows that $(I_A(v), I_A(w)) \in E_I(I(\texttt{sameAs}))$, so that $(I_A(v), I_A(w)) \in E_I(I(\texttt{subClassOf}))$.

Rule rdfp10 derives the triple p `subPropertyOf` q from p `type Property` and p `sameAs` q. We have $(I_A(p), I(\texttt{Property})) \in E_I(I(\texttt{type}))$, so that $I_A(p) \in CE_I(I(\texttt{Property})) = P_I$. Moreover, it follows that $(I_A(p), I_A(q)) \in E_I(I(\texttt{sameAs}))$, so that $(I_A(p), I_A(q)) \in E_I(I(\texttt{subPropertyOf}))$.

Rule rdfp11 is used to obtain $u'\,p\,v'$ from $u\,p\,v$, u `sameAs` u' and v `sameAs` v', where $u' \in U \cup B$. We have $(I_A(u), I_A(v)) \in E_I(I_A(p))$, $(I_A(u), I_A(u')) \in E_I(I(\texttt{sameAs}))$ and $(I_A(v), I_A(v')) \in E_I(I(\texttt{sameAs}))$. It is clear that $I_A(p) \in P_I$, so it follows that $(I_A(u'), I_A(v')) \in E_I(I_A(p))$.

Suppose that rule rdfp12a is used to obtain v `subClassOf` w from v `equivalentClass` w. Then, we have $(I_A(v), I_A(w)) \in E_I(I(\texttt{equivalentClass}))$, so that $(I_A(v), I_A(w)) \in E_I(I(\texttt{subClassOf}))$,

as desired. Suppose that rule rdfp12b is used to derive w `subClassOf` v from v `equivalentClass` w. Then we have again that $(I_A(v), I_A(w)) \in E_I(I(\texttt{equivalentClass}))$, so that $(I_A(w), I_A(v)) \in E_I(I(\texttt{subClassOf}))$. Suppose that rule rdfp12c is used to obtain v `equivalentClass` w from v `subClassOf` w and w `subClassOf` v. Then we have $(I_A(v), I_A(w)), (I_A(w), I_A(v)) \in E_I(I(\texttt{subClassOf}))$, so that $(I_A(v), I_A(w)) \in E_I(I(\texttt{equivalentClass}))$. The validity of rules rdfp13a, rdfp13b and rdfp13c is proved in the same way.

Suppose that rule rdfp14a is used to obtain u `type` v from v `hasValue` w, v `onProperty` p and $u\,p\,w$. Then, we have $(I_A(v), I_A(w)) \in E_I(I(\texttt{hasValue}))$, $(I_A(v), I_A(p)) \in E_I(I(\texttt{onProperty}))$ and $(I_A(u), I_A(w)) \in E_I(I_A(p))$, so that $I_A(u) \in CE_I(I_A(v))$ and $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$, as desired. Suppose that rule rdfp14bx is used to derive $u\,p\,w$ from v `hasValue` w, v `onProperty` p and u `type` v, where $p \in U \cup B$. Then, we have $(I_A(v), I_A(w)) \in E_I(I(\texttt{hasValue}))$, $(I_A(v), I_A(p)) \in E_I(I(\texttt{onProperty}))$ and $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$, so that $(I_A(u), I_A(w)) \in E_I(I_A(p))$, as desired.

Rule rdfp15 derives the triple u `type` v from v `someValuesFrom` w, v `onProperty` p, $u\,p\,x$ and x `type` w. We have $(I_A(v), I_A(w)) \in E_I(I(\texttt{someValuesFrom}))$, $(I_A(v), I_A(p)) \in E_I(I(\texttt{onProperty}))$, $(I_A(u), I_A(x)) \in E_I(I_A(p))$ and $(I_A(x), I_A(w)) \in E_I(I(\texttt{type}))$, so that $I_A(u) \in CE_I(I_A(v))$ and $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$, as required.

Rule rdfp16 is used to obtain x `type` w from v `allValuesFrom` w, v `onProperty` p, u `type` v and $u\,p\,x$, where $x \in U \cup B$. We have $(I_A(v), I_A(w)) \in E_I(I(\texttt{allValuesFrom}))$, $(I_A(v), I_A(p)) \in E_I(I(\texttt{onProperty}))$, $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$ and $(I_A(u), I_A(x)) \in E_I(I_A(p))$, so that $I_A(x) \in CE_I(I_A(w))$ and $(I_A(x), I_A(w)) \in E_I(I(\texttt{type}))$, as desired. □

5.14 Corollary. *Let D be a finite datatype map. The pD* entailment relation $S \models_{\mathrm{p}} G$ between finite sets S of finite generalized RDF graphs and finite generalized RDF graphs G is decidable. This problem is in NP, and in P if G is ground.*

Proof. This follows in the same way as in the case of D* entailment (see Corollary 4.13), noting in addition that P-clashes can be detected in polynomial time. □

5.15 Proposition. *Let D be a finite datatype map. The pD* entailment relation $S \models_{\mathrm{p}} G$ between finite sets S of finite generalized RDF graphs and finite generalized RDF graphs G is NP-complete.*

Proof. Two additions need to be made to the proof of Proposition 4.14. In the first part of the proof it is also assumed, of course, that the predicate p of the triples of G'' and H' is not in the P-vocabulary. With regard to the

last part of the proof it should be noted that $G' - G''$ contains a fourth triple containing p and that $G' - G''$ contains no further triples containing p; rule rdfp5a produces the triple p `sameAs` p. Therefore, it is still true that none of the triples in $G' - G''$ has p as predicate. □

A set S of generalized RDF graphs is called *pD* consistent* if there is a pD* interpretation that satisfies S.

5.16 Proposition. *Let D be a datatype map, S be a set of generalized RDF graphs, and H a partial pD* closure of $M(S)$. Then, S is pD* consistent if and only if H does not contain a P-clash or a D-clash. The problem to determine whether a finite set of finite generalized RDF graphs is pD* consistent is in P.*

Proof. These statements follow in a similar way to the corresponding statements for D* consistency (see Proposition 4.15), by using Lemma 5.11, the proof of Lemma 5.13 ($\Leftarrow$), and Lemma 5.8. □

6 pD*sv interpretations and pD*sv entailment

The pD* semantics considered in the preceding section includes half of OWL's iff condition for `someValuesFrom`, as is reflected by one entailment rule, rdfp15 (see Table 7). It is possible to reflect the complete OWL semantics for `someValuesFrom` with two entailment rules by using RDF's blank nodes (cf. 1.8). This is shown in this section with a completeness result for an extension of the pD* semantics. This extension is called the pD*sv semantics. We introduce a new entailment rule for `someValuesFrom`: rdf-svx (cf. 1.8). Unlike the P-entailment rules of the preceding section, this entailment rule introduces new blank nodes. We do not extend the decidability and complexity results for pD* entailment obtained in the preceding section to pD*sv entailment; since rule rdf-svx introduces blank nodes, the proofs of Lemmas 4.8 and 5.8 do not extend.

6.1 Definition (pD*sv interpretations). The pD*sv semantics extends the pD* semantics with OWL's iff condition for `someValuesFrom`. Given a datatype map D, a *pD*sv interpretation* of a vocabulary V is a pD* interpretation of V (see Definition 5.2) that satisfies the following condition: If $(a, b) \in E_I(I(\texttt{someValuesFrom}))$ and $(a, p) \in E_I(I(\texttt{onProperty}))$, then $x \in CE_I(a)$ if and only if there is a $y \in CE_I(b)$ such that $(x, y) \in E_I(p)$. Note that the if part of this condition is already satisfied by pD* interpretations (see Definition 5.2).

6.2 Definition (pD*sv entailment). Given a datatype map D, a set S of generalized RDF graphs *pD*sv entails* a generalized RDF graph G if each

pD*sv interpretation I that satisfies S also satisfies G.

It is clear that each OWL Full interpretation [14] is a pD*sv interpretation. It follows that if the set of generalized RDF graphs S pD*sv entails the generalized RDF graph G, then S OWL Full entails G (cf. Definition 5.5).

6.3 Definition (entailment rule rdf-svx). The entailment rule **rdf-svx** is defined as follows: if a generalized RDF graph G contains the triples v `someValuesFrom` w, v `onProperty` p and u `type` v, where $p \in U \cup B$, then add to G the two triples $u\,p\,b$ and b `type` w, where b is a new blank node. Note that entailment rule rdf-svx may introduce a triple with a blank node in predicate position when applied to ordinary RDF triples.

6.4 Definition (partial pD*sv closure). A *partial pD*sv closure* $G_{\mathrm{p}'K}$ of a generalized RDF graph G is defined in the same way as a partial pD* closure $G_{\mathrm{p},K}$ of G (see Definition 5.7), with the addition that in the last step entailment rule rdf-svx may also be applied to triples v `someValuesFrom` w, v `onProperty` p and u `type` v, when there is no x such that the triples $u\,p\,x$ and x `type` w are included. We shall see that this restricted form of application of entailment rule rdf-svx, which serves to restrict the growth of partial closures, is sufficient for determining pD*sv entailment.

6.5 Definition (pD*sv Herbrand interpretation). Given a datatype map D and a generalized RDF graph G, a *pD*sv Herbrand interpretation* $P'_K(G)$ is defined as a pD* Herbrand interpretation $P_K(G)$ (see Definition 5.9), the only differences being that G_{p} is replaced by $G_{\mathrm{p}'} = G_{\mathrm{p}'\{1,2,\ldots\}}$ and $G_{\mathrm{p},K}$ by $G_{\mathrm{p}'K}$.

6.6 pD*sv satisfaction lemma. *Let G be a generalized RDF graph and D a datatype map. If the partial pD*sv closure $G_{\mathrm{p}'K}$ of G does not contain a P-clash or a D-clash, then $P'_K(G)$ is a pD*sv interpretation that satisfies $G_{\mathrm{p}'K}$.*

Proof. In this proof, $P'_K(G)$ is denoted by I and $G_{\mathrm{p}'K}$ by H. We use the same function $A : bl(H) \to R_I$ as in the proofs of Lemmas 4.10 and 5.11. We only need to add to the proof of these lemmas the proof of the only if condition for `someValuesFrom`) on pD*sv interpretations. Suppose that $(a, b) \in E_I(I(\texttt{someValuesFrom}))$, $(a, p) \in E_I(I(\texttt{onProperty}))$ and $x \in CE_I(a)$, so that H contains the triples $sur(a)$ `someValuesFrom` $sur(b)$, $sur(a)$ `onProperty` $sur(p)$ and $sur(x)$ `type` $sur(a)$. By rule rdf-svx, H contains triples $sur(x)$ $sur(p)$ v and v `type` $sur(b)$. Since obviously $v \in U \cup B$, we have $v = sur(I_A(v))$ by Equation (28), so that H contains the triples $sur(x)$ $sur(p)$ $sur(I_A(v))$ and $sur(I_A(v))$ `type` $sur(b)$. This shows that $I_A(v) \in CE_I(b)$ and $(x, I_A(v)) \in E_I(p)$, which is sufficient to conclude the proof. □

6.7 pD*sv entailment lemma. *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Then, S pD*sv entails G if*

and only if there is a generalized RDF graph H that can be derived from $M(S)$ combined with RDF and RDFS axiomatic triples and D-axiomatic triples and P-axiomatic triples, by application of D entailment rules and P-entailment rules and entailment rule rdf-svx, and that either contains an instance of G as a subset or contains a P-clash or a D-clash.*

6.8 pD*sv entailment lemma (alternative statement). *Let D be a datatype map, S a set of generalized RDF graphs and G a generalized RDF graph. Let H be a partial pD*sv closure $M(S)_{\mathrm{p}'K}$ of $M(S)$ and suppose that $i \in K$ for each* `rdf:_`$i \in V(G)$. *Then, S pD*sv entails G if and only if either H contains an instance of G as a subset or H contains a P-clash or a D-clash.*

Proof. We only need to add to the proof of Lemmas 4.11, 4.12, 5.12 and 5.13 the proof of the validity of rule rdf-svx. The function $A : bl(H) \to R_I$ associated with the interpretation I in the inductive proof of soundness will now be defined by $A(b) = I(l)$ if b is a blank node allocated to the literal l by rule lg and by $A(b) = A'(b)$ if $b \in bl(M(S))$; see the proof of Lemma 4.12 for A'. For blank nodes b introduced by entailment rule rdf-svx, we define $A(b)$ in a recursive way. Rule rdf-svx derives the triples $u\,p\,b$ and b `type` w from v `someValuesFrom` w, v `onProperty` p and u `type` v, where $p \in U \cup B$. By the induction assumption we have $(I_A(v), I_A(w)) \in E_I(I(\texttt{someValuesFrom}))$, $(I_A(v), I_A(p)) \in E_I(I(\texttt{onProperty}))$ and $(I_A(u), I_A(v)) \in E_I(I(\texttt{type}))$. Since I is a pD*sv interpretation, it follows that $(I_A(u), y) \in E_I(I_A(p))$ for some $y \in CE_I(I_A(w))$. Taking $A(b)$ to be y, we get $(I_A(u), A(b)) \in E_I(I_A(p))$ and $A(b) \in CE_I(I_A(w))$, as desired. □

A set S of generalized RDF graphs is called *pD*sv consistent* if there is a pD*sv interpretation that satisfies S.

6.9 Proposition. *Let D be a datatype map, S be a set of generalized RDF graphs and H a partial pD*sv closure of $M(S)$. Then, S is pD*sv consistent if and only if H does not contain a P-clash or a D-clash.*

Proof. This is proved in a similar way to Proposition 5.16. □

7 Conclusion

In this paper we have proved that RDFS entailment is decidable. It has also been proved that the problem 'S RDFS-entails G', with S and G finite RDF graphs, is NP-complete, and in P if G is assumed to have no blank nodes. These results were obtained by refining the central RDFS completeness result (the RDFS entailment lemma) by using partial closure graphs which are finite and computable in polynomial time for finite RDF graphs. These

completeness, decidability and complexity results for RDFS entailment were extended to D* entailment and pD* entailment, which support reasoning with datatypes and a subset of the OWL vocabulary, respectively. For these extensions an 'if-semantics' was used, in line with the semantics of RDFS and weaker than the standard semantics of OWL. We extended the completeness result for pD* entailment to incorporate OWL's complete iff condition for `someValuesFrom`. We proved that consistency for the RDFS, D* and pD* semantics is in P, and that simple entailment is NP-complete. We have also shown that the standard set of entailment rules for RDFS is incomplete and that this can be corrected by allowing blank nodes in predicate position.

Acknowledgment. I am grateful to Jan Korst for his suggestion with regard to the proof of Proposition 2.13 and to Warner ten Kate, Jan Korst and the anonymous referees of [10] for their comments on earlier versions. The anonymous referees of this paper provided a valuable list of comments that have been used to improve this paper.

References

[1] F. Baader et al. (Eds.), *The Description Logic Handbook*, Cambridge University Press, 2003.

[2] T. Berners-Lee, J. Hendler, O. Lassila , The Semantic Web, Scientific American, May 2001.

[3] R. Cori, D. Lascar, *Mathematical Logic*, Parts I and II, Oxford University Press, 2000, 2001.

[4] M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York, 1979.

[5] J. Grant, D. Beckett (Eds.), RDF Test Cases, W3C Recommendation, 10 February 2004, `http://www.w3.org/TR/2004/REC-rdf-testcases-20040210/`

[6] C. Gutierrez, C. Hurtado, A.O. Mendelzon, Foundations of Semantic Web Databases, Proceedings of the 23rd Symposium on Principles of Database Systems (PODS2004), ACM, June 2004, pp. 95-106.

[7] P.R. Halmos, *Naive Set Theory*, Springer-Verlag, New York 1974.

[8] P. Hayes (Ed.), RDF Semantics, W3C Recommendation, 10 February 2004, `http://www.w3.org/TR/2004/REC-rdf-mt-20040210/`

[9] I. Horrocks, P.F. Patel-Schneider, Reducing OWL Entailment to Description Logic Satisfiability, Journal of Web Semantics, 1 (2004) 345-357.

[10] H. J. ter Horst, Extending the RDFS Entailment Lemma, in S.A. McIlraith et al. (Eds.), Proceedings of the 3rd International Semantic Web Conference (ISWC2004), November 2004, Springer, LNCS 3298, pp. 77-91.

[11] H.J. ter Horst, Semantic Web Ontologies and Entailment: Some Complexity Results, in W.F.J. Verhaegh et al. (Eds.), SOIA'04: Proceedings of the 2nd Philips Symposium on Intelligent Algorithms, pp. 59-71, Philips Research, Eindhoven, The Netherlands, December 2004.

[12] G. Klyne, J. Carroll (Eds.), Resource Description Framework (RDF): Concepts and Abstract Syntax, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[13] OWL (Web Ontology Language), W3C, http://www.w3.org/2004/OWL/

[14] P.F. Patel-Schneider, P. Hayes, I. Horrocks (Eds.), OWL Web Ontology Language Semantics and Abstract Syntax, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[15] RDF (Resource Description Framework), W3C, http://www.w3.org/RDF/

[16] RDF Data Access Working Group, W3C, http://www.w3.org/2001/sw/DataAccess/