RDFLog: It's like Datalog for RDF

François Bry[1], Tim Furche[1], Clemens Ley[2], Benedikt Linse[1], and Bruno Marnette[2]

[1] Institute for Informatics, University of Munich,
Oettingenstraße 67, D-80538 München, Germany
[2] Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford, OX1 3QD, England

Abstract. RDF data is set apart from relational or XML data by its support of rich existential information in the form of *blank nodes*. Where SQL null values are scoped over a single statement, blank nodes in RDF can span over any number of statements and thus can be seen as existentially quantified variables.
For RDF querying blank node querying is considered in most query languages, but blank node construction, i.e., the introduction of new blank nodes has been mostly ignored (e.g., in Triple) or treated in a very limited form (e.g., in SPARQL). In this paper, we classify three kinds of blank node in RDF query languages and introduce the recursive, rule-based RDF query language RDFLog. RDFLog is the first RDF query languages with full arbitrary quantifier alternation: blank nodes may occur in the scope of all, some, or none of the universal variables of a rule. RDFLog is also aware of important RDF features such as the distinction between blank nodes, literals and URIs or the RDFS vocabulary.

1 Introduction

Access to data in a machine-processable, domain-independent manner plays a central role in the future growth of the Internet. Information on legislative proceedings, census data, scientific experiments and databases, as well as the data gathered by social network applications is now accessible in form of RDF data. The Resource Description Framework (RDF) is a data format for the Web with a formal semantics that is achieving considerable popularity. Compared to relational databases, RDF is mostly distinguished by (1) a specialization to ternary statements or "triples" relating a subject, via a predicate, to an object, (2) the presence of blank nodes that allow statements where subject or object are unknown, and (3) specific semantics for a small, predefined vocabulary (RDF Schema, or RDFS) reminiscent of an object-oriented type system.

With the staggering amount of data available in RDF form on the Web, the second indispensable ingredient becomes the easy selection and processing of RDF data. For that purpose, a large number of RDF query languages (see [1] for a recent survey) has been proposed. In this paper, we add a further exemplar: RDFLog extends datalog to support the distinguishing features of RDF such as blank nodes and the logical core [2] of the RDFS vocabulary. In

RDFLog, Blank nodes can be constructed by existentially quantified variables in rule heads. RDFLog allows *full alternation* between existential and universal quantifiers in a rule. This sharply contrasts with previous approaches to rule-based query languages that either do not support blank nodes (in rule heads) at all [3,4], or only a limited form of quantifier alternation [5,6,7].

To illustrate the benefits of full quantifier alternation, imagine an information system about university courses. We distinguish three types of rules with existential quantifiers (and thus blank nodes) based on the alternation of universal and existential quantifiers:

(1) "Someone knows each professor" can be represented in RDFLog as

$$\exists stu \forall prof \, ((prof, \mathsf{rdf{:}type}, \mathsf{uni{:}professor}) \rightarrow (stu, \mathsf{foaf{:}knows}, prof)) \qquad (1)$$

We call such rules $\exists\forall$ rules. Some approaches such as [5] are limited to rules of this form.

(2) Imagine, that we would like to state that each lecture must be "practiced" by another course (such as a tutorial or practice lab) without knowing more about that course. This statement can not be expressed by $\exists\forall$ rules. In RDFLog it can be represented as

$$\forall lec \exists crs \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \rightarrow (crs, \mathsf{uni{:}practices}, lec)\big) \qquad (2)$$

Such rules are referred to as $\forall\exists$ rules. Recent proposals for rule extensions to SPARQL are limited to this form, if they consider blank nodes in rule heads at all. The reason is that in SPARQL `CONSTRUCT` patterns a fresh blank node is constructed for each binding of the universal variables (cf. Section 10.2.1 in [8]).

(3) To the best of our knowledge, RDFLog is the first RDF query language that supports the third kind of rules, where quantifiers are allowed to alternate freely: This allows to express statements such as, for each lecture there is a course that "practices" that lecture and is attended by all students attending the lecture. This is represented in RDFLog as

$$\forall lec \exists crs \forall stu \big((lec, \mathsf{rdf{:}type}, \mathsf{uni{:}lecture}) \wedge (stu, \mathsf{uni{:}attends}, lec) \rightarrow \\ (crs, \mathsf{uni{:}practices}, lec) \wedge (stu, \mathsf{uni{:}attends}, crs)\big) \qquad (3)$$

In addition to flexible support for existential information through full quantifier alternation, RDFLog captures the essentials of RDF through two further characteristics: First, RDFLog is a closed RDF query language, i.e., the answer to an RDFLog program is again an RDF graph. Second, RDFLog can express the logical core of the RDFS semantics (ρdf from [2]).

In particular, we follow RDF in allowing blank nodes not in predicate position for answers (as well as literals only in object position). We show that these limitations make the traditional approach of defining a closed semantics for a rule based query language as initial models unpractical. Nonetheless, we show how a closed semantics of a rule based query language for RDF can be defined that captures the consequences of the program under RDF entailment.

Contributions. The paper is organised along the following contributions:

1. A rule-based RDF query language combining *recursion and free quantifier alternation*, called RDFLog (Section 3) is introduced.
2. The closed semantics of RDFLog (with or without a core fragment of the RDFS semantics) is introduced in terms of RDF entailment. (Section 3.3).
3. We show how this semantics can be implemented by a *reduction to the evaluation of a standard logic program* without existential quantifiers. (Section 4).
4. The experimental evaluation of a basic prototype shows that the reduction to standard logic programming easily competes with existing specialized RDF query engines even when considering only the restricted fragment of RDFLog equivalent to SPARQL (Section 5).

2 Preliminaries

2.1 Syntax and Semantics of RDF

In this paper, we adopt the notions of RDF vocabulary, RDF graph, (simple) RDF interpretation, and RDF entailment from [10].

Definition 1 (RDF Graph [10]). *An* RDF vocabulary V *consists of two disjoint sets called* URIs U *and* literals L*. The* blank nodes B *is a set disjoint from* U *and* L*. An* RDF graph *is a set of RDF triples where an* RDF triple *is an element of* $(\mathsf{U} \cup \mathsf{B}) \times \mathsf{U} \times (\mathsf{U} \cup \mathsf{L} \cup \mathsf{B})$*. If* $t = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ *is an RDF triple then* s *is the* subject*,* p *is the* predicate*, and* o *is the* object *of* t*.*

The set L of literals consists of three subsets, *plain literals*, *typed literals* and *literals with language tags*. In this work we consider only plain literals (and thus drop IL, the interpretation function for typed literals, see Section 1.3 in [10], in the following definitions).

Definition 2 (RDF Interpretation [10]). *An* interpretation I *of an RDF vocabulary* $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ *is a tuple* $(\mathsf{IR}, \mathsf{LV}, \mathsf{IP}, \mathsf{IEXT}, \mathsf{IS})$ *where* IR *is a non-empty set of* resources *such that* $\mathsf{L} \subseteq \mathsf{LV} \subseteq \mathsf{IR}$*,* IP *is a set of* properties *and* $\mathsf{IEXT} : \mathsf{IP} \rightarrow 2^{\mathsf{IR} \times \mathsf{IR}}$*, and* $\mathsf{IS} : \mathsf{U} \rightarrow \mathsf{IR} \cup \mathsf{IP}$ *are mappings.*

Note that as IR and IP are not necessarily disjoint a same URI can be used both as a resource and a property. RDF interpretations are used to assign a truth value to an RDF graph.

RDF assigns a special meaning to a predefined vocabulary, called RDFS vocabulary. For example it is required that $\mathsf{IEXT}(\mathsf{IP}(\mathsf{rdfs:subPropertyOf}))$ is transitive and reflexive. The formulation of theses constraints on RDF interpretation makes use of a notion of a *class*. We have omitted this notion in the definition above for simplicity. The logical core of RDFS has been identified in [2], denoted as ρdf. An RDF interpretation I is a *ρdf interpretation* if I satisfied the constraints specified in Definition 3 in [2].

Definition 3 (Interpretation of an RDF Graph [10]). *Let* I *be the RDF (ρdf) interpretation* $(\mathsf{IR}, \mathsf{LV}, \mathsf{IP}, \mathsf{IEXT}, \mathsf{IS})$ *and* $\mathsf{A} : \mathsf{B} \rightarrow \mathsf{IR}$ *a mapping. Then*

$[\mathsf{I}+\mathsf{A}](\mathsf{e}) = \mathsf{a}$ *if* e *is the literal* a*,* $[\mathsf{I}+\mathsf{A}](\mathsf{e}) = \mathsf{IS}(\mathsf{e})$ *if* e *is a URI,* $[\mathsf{I}+\mathsf{A}](\mathsf{e}) = \mathsf{A}(\mathsf{e})$ *if* e *is a blank node, and* $[\mathsf{I}+\mathsf{A}](\mathsf{e}) = \text{true}$ *if* $\mathsf{e} = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ *is an RDF triple over* V*,* $\mathsf{I}(\mathsf{p}) \in \mathsf{IP}$ *and* $(\mathsf{I}(\mathsf{s}), \mathsf{I}(\mathsf{o})) \in \mathsf{IEXT}(\mathsf{I}(\mathsf{p}))$*. Finally* $\mathsf{I}(\mathsf{g}) = \text{true}$ *if there is a mapping* $\mathsf{A} : \mathsf{B} \to \mathsf{IR}$ *such that* $[\mathsf{I}+\mathsf{A}](t) = \text{true}$ *for all RDF triples* $\mathsf{t} \in \mathsf{g}$*.*

The semantics of RDF is completed by the notion of entailment: An RDF graph g *RDF-entails* (*ρdf-entails*) an RDF graph h if for all RDF (*ρdf*) interpretations I, $\mathsf{I}(\mathsf{h}) = \text{true}$ if $\mathsf{I}(\mathsf{g}) = \text{true}$ [10].

2.2 Logic and Logic Programming

We recall some concepts from logic and logic programming used in the discussion of RDFLog in Section 3 and 4.

Formulas and *terms* over an alphabet Σ are defined as in first order logic. In addition we consider infinite formulas: if Φ is a countably infinite set of formulas then $\bigwedge(\Phi)$ is a formula and if $\bar{x} = x_1, x_2, \ldots$ is a countably infinite sequence of variables and φ is a formula then $\exists\bar{x}(\varphi)$ is a formula. We write $\varphi(\bar{x})$ to indicate that the free variables of a formula φ are among $\bar{x} = x_1, \ldots, x_n$.

To avoid confusion with RDF interpretations, we use the notion of structure (rather than first-order interpretation). A *structure* A over an alphabet Σ is a tuple $(D, \mathit{Fun}, \mathit{Rel})$ where D is a set called the domain of A, *Fun* is a set containing a function f^A for every function symbol f in Σ and *Rel* is a set containing a relation R^A for every relation symbol R in Σ. A structure A over Σ is a *Herbrand structure* if D is the set of ground terms over Σ and every n-ary function $f^A : D^n \to D$ in *Fun* is defined by $f^A(\bar{t}) = f(\bar{t})$.

The *satisfaction relation* $\models$ between structures and formulas and the notion of a *model* is defined as in first order logic, with the obvious extension for infinite conjunction $\bigwedge$ and infinite existential quantification $\exists\bar{x}$. We write $A \models \varphi(d_1, \ldots, d_n)$ to indicate that $\varphi(x_1, \ldots, x_n)$ is true in A if the free variables $x_1, \ldots, x_n$ in $\varphi(x_1, \ldots, x_n)$ are interpreted by the domain elements $d_1, \ldots, d_n$ respectively. *Entailment* is defined as usual and denoted by $\models$.

A *logic program* is a finite set of sentences $\forall\bar{x}\,(a_1(\bar{x}) \wedge \ldots \wedge a_n(\bar{x}) \to a(\bar{x}))$ where $a_1(\bar{x}), \ldots, a_n(\bar{x})$ and $a(\bar{x})$ are atoms. A logic program P has a unique minimal Herbrand model denoted by M_P. A *Datalog* program is a logic program **containing** no function symbols of arity greater than zero (i.e. constants).

3 Syntax and Semantics of RDFLog

3.1 The RDFLog Data Model

To make results from databases and logic programming accessible for RDF querying, we show that the semantics of RDF can be defined in terms of standard logic. In particular we show that RDF graphs can be translated to formulas so that logical entailment coincides with RDF entailment.

For any RDF vocabulary $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ we define the alphabet $\Sigma_{\mathsf{V}} = \mathsf{U} \cup \mathsf{L} \cup \{T\}$ where U and L are constant symbols and T is an arbitrary ternary relation symbol.

Definition 4 (Canonical Formula of an RDF Graph). *Let* $\mathsf{g} = \{\mathsf{t}_1, \dots, \mathsf{t}_n\}$ *be an RDF graph over* V*. The* canonical formula *of* g *is the formula* $\varphi_{\mathsf{g}} := \exists \bar{x}\, (\psi_1(\bar{x}) \wedge \dots \wedge \psi_n(\bar{x}))$ *over* Σ_{V} *and variables from* B *where* $\psi_i = T(\mathsf{s}, \mathsf{p}, \mathsf{o})$ *if* $\mathsf{t}_i = (\mathsf{s}, \mathsf{p}, \mathsf{o})$ *and* $\bar{x}$ *is the set of blank nodes occurring in* g*.*

In [2] a sound and complete deductive system for ρdf has been presented. It is easy to see that this deductive system corresponds to a finite set of Datalog rules $\Phi^{\rho df}$.

Proposition 1. *Let* g, h *be RDF graphs and* φ_{g}, φ_{h} *their canonical formulas. Then* g *RDF-entails* h *iff* $\varphi_{\mathsf{g}} \models \varphi_{\mathsf{h}}$ *and* g ρdf*-entails* h *iff* $\varphi_{\mathsf{g}} \wedge \Phi^{\rho df} \models \varphi_{\mathsf{h}}$.[1]

3.2 RDFLog Syntax

Definition 5 (Syntax of RDFLog Programs). *Let* $\mathsf{V} = (\mathsf{U}, \mathsf{L})$ *be an RDF vocabulary and Var a set of variables. An* RDFLog atom *over* V *is an atom* $T(t_1, t_2, t_3)$ *where* $t_1, t_2 \in (\mathsf{U} \cup \mathit{Var})$ *and* $t_3 \in (\mathsf{U} \cup \mathsf{L} \cup \mathit{Var})$*. An* RDFLog rule *over* V *is a formula*

$$\forall \bar{x}_1 \exists \bar{y}_1 \dots \forall \bar{x}_n \exists \bar{y}_n \, (\mathit{body}(\bar{x}) \rightarrow \mathit{head}(\bar{x}, \bar{y}))$$

over Σ_{V} *and Var where* $\bar{x} = \bar{x}_1, \dots, \bar{x}_n$ *and* $\bar{y} = \bar{y}_1, \dots, \bar{y}_n$ *are finite sequences from Var and* $\mathit{body}(\bar{x})$ *and* $\mathit{head}(\bar{x}, \bar{y})$ *are finite conjunctions of RDFLog atoms. In addition we require that RDFLog rules are* range restricted*: if* $x \in \mathit{Var}(\mathit{head})$ *is universal or there is an existential* $y \in \mathit{Var}(\mathit{head})$ *such that* y *is in the scopeof* x*, then* $x \in \mathit{Var}(\mathit{body})$*. An* RDFLog program *over* V *is a finite set of RDFLog rules over* V*.*

Observe that any finite RDF graph $\mathsf{g} = \{t_1, \dots, t_n\}$ with blank nodes $\bar{x}$ can be encoded into the RDFLog rule $\exists \bar{x}\, (\mathit{true} \rightarrow t_1 \wedge \dots \wedge t_n)$ where *true* denotes the empty conjunction. As it makes the notation simpler we always assume that the input RDF graph is encoded into a rule in the RDFLog program. As there is only one predicate symbol (T) in an RDFLog program it is usually omitted.

3.3 RDFLog Semantics

It is not generally agreed upon what a the semantics of a rule based RDF query language should be if existential variables are allowed in the head. In contrast, it is agreed that the semantics of a logic program with only universally quantified variables is its minimal Herbrand model.

The following RDFLog program illustrates why it is problematic to define the semantics of an RDF query language directly in terms of models. Let the *canonical structure* A_{g} of an RDF graph g be the structure over the domain of URIs, literals and blank nodes where (t_1, t_2, t_3) is true in A_{g} iff (t_1, t_2, t_3) is

[1] For proofs of theorems, lemmas, and proposition see the appendix of the online version [11].

an RDF triple in g. As (2) is a fact in P and (1) is a rule in P, any canonical structure of an RDF graph that is a model of P must contain the triple $(\mathsf{'Logic'}, \mathsf{uni{:}located\_in}, \mathsf{\_{:}b})$ for some blank node $\mathsf{\_{:}b}$. Since this triple contains a literal in the subject position, it is not an RDF triple. This illustrates that P has no model that is the canonical structures of an RDF graph. Even if literals in subject position are allowed (as in SPARQL), a similar argument can be made with blank nodes in predicate position.

$$\begin{aligned}
P = \big\{&\forall sem \exists rm \forall stu \big((stu, \mathsf{uni{:}attends}, sem)\\
&\quad \rightarrow (sem, \mathsf{uni{:}located\_in}, rm) \wedge (stu, \mathsf{uni{:}knows}, rm)\big), && (1)\\
&\quad true \rightarrow (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{'Logic'}) \wedge (\mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF})\big\} && (2)\\
[\![P]\!] \ni \big\{&(\mathsf{\_{:}b3}, \mathsf{uni{:}located\_in}, \mathsf{\_{:}b1}), (\mathsf{uni{:}julie}, \mathsf{uni{:}knows}, \mathsf{\_{:}b1}),\\
&(\mathsf{uni{:}RDF}, \mathsf{uni{:}located\_in}, \mathsf{\_{:}b2}), (\mathsf{uni{:}john}, \mathsf{uni{:}knows}, \mathsf{\_{:}b2}),\\
&(\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{`Logic'}), (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b3}),\\
&(\mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF})\big\}
\end{aligned}$$

We deal with this problem by defining the semantics of RDFLog in terms of RDF entailment. More precisely we define the semantics of an RDFLog program P to be the set of all RDF graphs g that entail exactly the same RDF graphs as P (and satisfying in particular $P \models g$).

Definition 6 (Denotational Semantics of RDFLog). *Let P be an RDFLog program and* RDF *the set of RDF graphs. The* denotational semantics $[\![P]\!]$ *of P is the set* $[\![P]\!] := \{\mathsf{g} \in \mathsf{RDF} \mid \forall \mathsf{h} \in \mathsf{RDF}\, (P \models \varphi_{\mathsf{h}}$ *iff* $\varphi_{\mathsf{g}} \models \varphi_{\mathsf{h}})\}$ *where φ_{g} and φ_{h} are the canonical formulas of* g *and* h *respectively.*

Observe that the semantics of an RDFLog program is a infinite set of possibly infinite RDF graphs. As we formalised RDF graphs as formulas, we have to consider the special kind of infinite formulas defined in section 2.2. Nonetheless it is immediate from the definition that the RDF graphs in $[\![P]\!]$ form an equivalence class under RDF entailment. Therefore any element of $[\![P]\!]$ characterizes the infinite set $[\![P]\!]$. In the next section we show how such a representative can be computed.

Observe that $\Phi^{\rho df}$ encoded in RDFLog. Therefore it is up to the programmer to enclose $\Phi^{\rho df}$ into P if the semantics of P is supposed to be aware of the ρdf vocabulary.

4 Evaluation

The goal of this section is to show how the evaluation of an RDFLog program P can be done by first translating P into a logic program $s(P)$, using the well-studied notion of Skolemisation, and then evaluate this program $s(P)$ using standard technology. Two post processing steps (Unskolemisation and RDF normalization) make sure that the result is an RDF graph in the denotational semantics of P. After defining precisely each of the key steps of the operational semantics in Section 4.1, we show in Section 4.2 that the operational semantics achieves its goal as it is consistent with the denotational semantics of RDFLog.

4.1 Operational Semantics of RDFLog

Definition 7 (Skolemisation). *Let Σ and Γ be disjoint alphabets, $\varphi = \forall \bar{x} \exists y(\psi)$ a formula over $\Sigma \cup \Gamma$ and $f \in \Gamma$. A Γ-*Skolemisation step *s_f maps φ to $s_f(\varphi) := \forall \bar{x} \psi\{y \leftarrow f(\bar{x})\}$. A Γ-*Skolemisation *s is a composition $s_{f_1} \circ \ldots \circ s_{f_n}$ of Γ-Skolemisation steps such that f_i does not occur in $s_{f_{i+1}} \circ \ldots \circ s_{f_n}(\varphi)$ and $s(\varphi)$ contains no existential variables. The definition of a Skolemisation is extended to sets in the usual way.*

The Skolemisation of an RDFLog program P is equivalent to a range restricted logic program, which we denote by $s(P)$. Any logic programming engine can compute the minimal Herbrand model $M_{s(P)}$ of $s(P)$. The following logic program is the Skolemisation $s(P)$ of the RDFLog program P from Section 3.3 where s replaces the existential variable rm in P by the term $s_{rm}(sem)$.

$$
\begin{aligned}
s(P) = \big\{ & \forall sem \forall stu \big((stu, \mathsf{uni{:}attends}, sem) \\
& \quad \to (sem, \mathsf{uni{:}located\_in}, s_{rm}(sem)) \land (stu, \mathsf{uni{:}knows}, s_{rm}(sem)) \big), \\
& \quad true \to (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{'Logic'}) \land \mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF}) \big\}
\end{aligned}
$$

$$
\begin{aligned}
\varphi_{M_{s(P)}} = {} & (\mathsf{`Logic'}, \mathsf{uni{:}located\_in}, s_{rm}(\mathsf{`Logic'})) \land (\mathsf{uni{:}julie}, \mathsf{uni{:}knows}, s_{rm}(\mathsf{`Logic'})) \\
& \land (\mathsf{uni{:}RDF}, \mathsf{uni{:}located\_in}, s_{rm}(\mathsf{uni{:}RDF})) \land (\mathsf{uni{:}john}, \mathsf{uni{:}knows}, s_{rm}(\mathsf{uni{:}RDF})) \\
& \land (\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{`Logic'}) \land (\mathsf{uni{:}john}, \mathsf{uni{:}attends}, \mathsf{uni{:}RDF})
\end{aligned}
$$

We define $\varphi_{M_{S(P)}}$ to be the conjunction of all ground atoms that are true in $M_{s(P)}$. However, $\varphi_{M_{S(P)}}$ might not be the canonical formula of an element of $[\![P]\!]$ for two reasons. First, the example shows that $\varphi_{M_{S(P)}}$ might contain atoms with skolem terms, such as $(\mathsf{uni{:}RDF}, \mathsf{uni{:}located\_in}, s_{rm}(\mathsf{uni{:}RDF}))$, which are not entailed by P. Second, $\varphi_{M_{S(P)}}$ can contain atoms that contain literals in subject or predicate position and blank nodes in predicate position. In the example the atom $(\mathsf{`Logic'}, \mathsf{uni{:}located\_in}, s_{rm}(\mathsf{`Logic'}))$ contains the literal $\mathsf{`Logic'}$ in subject position.

We can avoid the first problem by "undoing" the Skolemisation: replacing each Skolem term in $\varphi_{M_{S(P)}}$ by a fresh, distinct blank node. We formalise this operation as the inverse of a Skolemisation called *Unskolemisation*.

Definition 8 (Unskolemisation). *Let Σ and Γ be disjoint alphabets and φ a ground, possibly infinite, and quantifier free formula over $\Sigma \cup \Gamma$. Let $\bar{t}$ be the sequence of all ground terms $f(\bar{u})$ where f is in Γ and $\bar{u}$ is a sequence of terms over $\Sigma \cup \Gamma$. Then the Γ-*Unskolemisation *u maps φ to $u(\varphi) := \exists \bar{x} \left(\varphi\{\bar{t} \leftarrow \bar{x}\}\right)$. where $\bar{x}$ is a sequence of fresh variables.*

To address the second issue, we remove all triples with literals or blank nodes in predicate position (no RDF graph may contain such a triple or any triple entailed by it). In addition we remove each triple t that contains a literal

l in object position and add two triples t_1 and t_2 where t_1 is obtained from t by replacing an occurrence of a literal l in subject position by a fresh blank node b_l and t_2 is obtained from t by replacing all occurrences of l by b_l.

This is necessary to preserve information about the identity of domain elements that are denoted by blank nodes. For example observe that the RDF graph $\{(\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b}), (\mathsf{\_{:}b}, \mathsf{uni{:}located\_in}, s_{rm}(\text{‘}\mathsf{Logic}\text{’}))\}$ follows from the RDFLog program P in Section 3.3. To maintain this information we need to insert the triple the triple $(\mathsf{uni{:}julie}, \mathsf{uni{:}attends}, \mathsf{\_{:}b3})$ into $[\![P]\!]$. We formalise this step by defining the normalisation operator.

Definition 9 (Normalisation Operator). *Let φ be a formula of the form $\exists \bar{x}\, (a_1(\bar{x}) \wedge \ldots \wedge a_n(\bar{x}))$ where each $a_i(\bar{x}) = T(t_1, t_2, t_3)$ for some $t_1, t_2, t_3 \in (\mathsf{U} \cup \mathsf{B} \cup \mathsf{L})$. Let $\mathsf{L}' \subseteq \mathsf{L}$ be the set of literals that occur in the first argument of an atom in φ. We define $\mu : \mathsf{U} \cup \mathsf{B} \cup \mathsf{L} \to \mathsf{U} \cup \mathsf{B} \cup \mathsf{L}$ to be the injection such that $\mu(t) = b$ for some fresh blank node b (not in φ) if $t \in \mathsf{L}'$ and $\mu(t) = t$ otherwise. Then $\Pi(\varphi) = \{\Pi(a_1(\bar{x})), \ldots \Pi(a_n(\bar{x}))\}$ and*

$$\Pi(T(t_1,t_2,t_3)) = \begin{cases} \top & \text{if } t_2 \in \mathsf{B} \cup \mathsf{L} \\ (\mu(t_1), t_2, t_3) \wedge (\mu(t_1), t_2, \mu(t_3)) & \text{otherwise} \end{cases}$$

The normalisation operator ensures that, though intermediary triples may contain blank nodes in predicate position (see [12] for examples where this is useful), the final answer of an RDFLog program never contains such triples.
Armed with these notions of Skolemisation, Unskolemisation and Normalisation, we finally define the operational semantics of RDFLog as follow.

Definition 10 (Operational Semantics of RDFLog). *Let P be an RDFLog program over Σ, s a Γ-Skolemisation for P, and u an Γ-Unskolemisation. Then the* operational semantics *of P is $[P] := \Pi\left(u(\varphi_{M_{s(P)}})\right)$ where $\varphi_{M_{S(P)}}$ is as defined above: the conjunction of all ground atoms that are true in the minimal Herbrand model of $s(P)$.*

4.2 Properties of the Operational Semantics

Even though we do not require that elements of the denotational semantics $[\![P]\!]$ of an RDFLog program P are models of P it holds that $u(\varphi_{M_{s(P)}})$ has a canonical structure that is not only a model of P but even a universal model [9]. Thus if we allow literals in subject position and blank nodes in subject or predicate position, we can omit Π from the operational semantics and compute a model of P.

To formulate this more precisely, we define an *extended Herbrand structure A* over alphabet Σ and variables Var as a structure (D, Rel, Fun) where D is the set of (possibly non-ground) terms over Σ and Var, and every function f^A is defined by $f^A(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$. We extend the definition of Unskolemisation from formulas to extended Herbrand structures: if u is an Unskolemisation that replaces $\bar{t}$ by $\bar{x}$ then $u(M)$ is the extended Herbrand structure obtained from M by renaming the domain elements $\bar{t}$ by $\bar{x}$.

Lemma 1. *Let P be an RDFLog program, $A_P = u(M_{s(P)})$ and $\varphi_P = u\left(\varphi_{M_{s(P)}}\right)$. Then $A_P \models P$ and $P \models \varphi_P$.*

Intuitively, $A_P \models P$ means that φ_P captures all the information in P and $P \models \varphi_P$ means that it does not assert anything that is not asserted by P. From these two key observations, we can prove that the operational semantics of RDFLog is both sound and complete with respect to the denotational semantics.

Theorem 1. *Let P be an RDFLog program. Then $[P] \in [\![P]\!]$.*

5 Experimental Evaluation

The reduction of RDFLog to standard logic programs (Section 4) allows for a direct implementation of RDFLog on top of any logic programming or database engine that supports value invention and recursion. In the following, we we compare experimentally the performance of a very simple prototype based on that principle with two of the more common SPARQL implementations. Our implementation of RDFLog uses a combination of Perl pre- and post-filters for Skolemisation, Unskolemisation, and normalisation of RDFLog programs and XSB Prolog to evaluate the Skolemised programs.

We compare our implementation with the ARQ SPARQL processor of Jena (Version 2.1) and the SPARQL engine provided by the Sesame RDF Framework. For Sesame, we choose the main-memory store as it is "by far the fastest type of repository that can be used" according to Sesame's authors. With this store, Sesame becomes a main-memory, ad-hoc query engine just like RDFLog and ARQ. As common for ad-hoc queries we measure overall execution time including both loading of the RDF data and execution of the SPARQL or RDFLog query.

In the experiments we evaluate three different queries against an RDF graph consisting of Wikipedia data. The experiments have been carried out on a Intel Pentium M Dual-Core with 1.86 GHz, 1 MB cache and 2 GB main memory. For each setting, the running time is averaged over 25 runs. We compare the following rules:

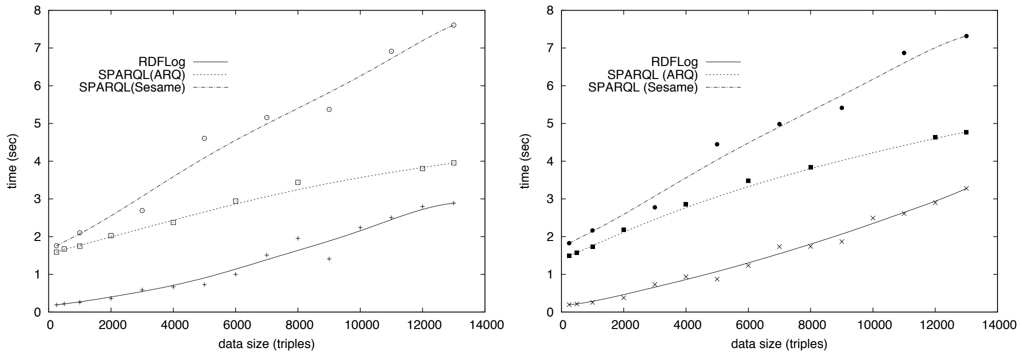
- *Rule 1:* $\forall x \forall y\, ((x, \mathsf{wiki{:}internalLink}, y) \rightarrow (x, \mathsf{test{:}connected}, y))$
- *Rule 2:* $\forall x \forall y \exists z\, ((x, \mathsf{wiki{:}internalLink}, y) \rightarrow (x, \mathsf{test{:}connected}, z))$

Figure 1 compares the performance of RDFLog with that of ARQ and Sesame for rule 1 and rule 2 (we omit rule 3 as it is not expressible in SPARQL). Despite its light-weight, ad-hoc implementation, RDFLog outperforms ARQ and Sesame in this setting. The figures show moreover that also for ARQ and Sesame, blank node construction does not bear any significant additional computational effort.

6 Conclusion

Blank nodes are one of RDF's distinguishing features. Yet they have been entirely neglected or treated only in a limit fashion in previous approaches to RDF

Fig. 1 Performance comparison on rule 1 (left) and on rule 2 (right)



querying. With RDFLog we propose a simple, yet comprehensive extension of Datalog that covers all aspects of blank node *construction* that arise when combining RDF with rule. We show that such an extension, including the restrictions of RDF wrt. blank node occurrence can be treated in a semantics based purely on entailment. Furthermore, RDFLog easily incorporates (the logical core of) RDFS. This allows us to view RDFLog as a convenient vessel for classifying and comparing RDF query languages, similar to the role of Datalog for relational databases. In particular, we identify four classes of blank node support in a rule based RDF query language: no support, in the scope of *no* universal variable, in the scope of *all* universal variables, or arbitrarily alternating with universal variables. Existing approaches fall in one of the three first classes, with RDFLog the first instance of the forth class.

Though RDFLog is primarily designed as a logical foundation for RDF query languages, we also show that it is easily implemented on top of existing logic programming technology and that such an approach actually compares very well with existing SPARQL engines.

Acknowledgements. We would like to thank Michael Benedikt for the fruitful discussions on RDFLog and the help with this article.

References

1. Furche, T., Linse, B., Bry, F., Plexousakis, D., Gottlob, G.: RDF Querying: Language Constructs and Evaluation Methods Compared. In: Tutorial Lectures Int'l. Summer School 'Reasoning Web'. Volume 4126 of Lecture Notes in Computer Science., Springer Verlag (2006) 1–52
2. Muñoz, S., Pérez, J., Gutierrez, C.: Minimal Deductive Systems for RDF. In: Proc. European Semantic Web Conf. (ESWC). Volume 4519 of Lecture Notes in Computer Science., Springer Verlag (2007) 53–67
3. Polleres, A.: From SPARQL to Rules (and Back). In: Proc. Int'l. World Wide Web Conf. (WWW), New York, NY, USA, ACM (2007) 787–796

4. Sintek, M., Decker, S.: Triple—a Query, Inference, and Transformation Language for the Semantic Web. In: Proc. Int'l. Semantic Web Conf. (ISWC). (2002)
5. Yang, G., Kifer, M.: Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. Journal of Data Semantics **1** (2003) 69–97
6. Schenk, S., Staab, S.: Networked Graphs: a Declarative Mechanism for SPARQL Rules, SPARQL Views and RDF Data Integration on the Web. In: Proc. Int'l. World Wide Web Conf. (WWW), New York, NY, USA, ACM (2008) 585–594
7. Gutierrez, C., Hurtado, C., Mendelzon, A.O.: Foundations of Semantic Web Databases. In: Proc. ACM Symp. on Principles of Database Systems (PODS), New York, NY, USA, ACM Press (2004) 95–106
8. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. Proposed recommendation, W3C (2007)
9. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. (2003) 207–224
10. Hayes, P., McBride, B.: RDF Semantics. Recommendation, W3C (2004)
11. Bry, F., Furche, T., Ley, C., Linse, B., Marnette, B.: RDFLog: It's like Datalog for RDF. Technical Report PMS-FB-2008-01, University of Munich (2008) `http://rdflog.com/publications/bry-rdflog-full.pdf`.
12. ter Horst, H.J.: Completeness, Decidability and Complexity of Entailment for RDF Schema and a Semantic Extension Involving the OWL Vocabulary. Web Semantics: Science, Services and Agents on the World Wide Web **3** (2005)

A Properties of the Operational Semantics Proof of Lemma 1 and Theorem 1

In the proof of Lemma 1 we often make use of the Substitution lemma, which we state here without proof.

Lemma 2 (Substitution Lemma). *Let φ be a sentence and M a structure. Then*

$$M \models \varphi \quad \textit{iff} \quad M, d \models \psi(x)$$

if $\psi = \varphi\{t \leftarrow x\}$ and M interpretes t as d.

We now recall some well known results about Skolemisation. Symmetric proofs show the following properties the Unskolemisation.

Lemma 3 (Skolemisation Lemma). *Let Σ, Γ and Π be disjoint alphabets and φ a finite formula over $\Sigma \cup \Gamma$. Let s be a Π-Skolemisation for φ, u an Γ-Unskolemisation for φ. Then*

- $\varphi \models u(\varphi)$
- $s(\varphi) \models \varphi$.
- *$u(\varphi)$ is satisfiable iff φ is satisfiable.*
- *φ is satisfiable iff $s(\varphi)$ is satisfiable.*

Corollary 1 (of the Skolemisation Lemma). *Let φ be a finite formula over $\Sigma \cup \Gamma$ and u an Γ-Unskolemisation for φ. If S is a model of $u(\varphi)$ over Σ then there exists an extension T of S on Γ which is a model of φ.*

The next Lemma is the central step to show that every RDF graph which is entailed by the operational semantics of an RDFLog program P is entailed by P.

Lemma 4. *Let φ and ψ be formulas over $\Sigma \cup \Gamma$ where φ is finite and ψ is possibly infinite and ground. Let u be an Γ-Unskolemisation for φ. Then*

$$\varphi \models \psi \quad \textit{implies that} \quad u(\varphi) \models u(\psi).$$

Proof. Let S be a model over Σ of $u(\varphi)$. As φ is finite, by Corollary 1 there is an extension T of S on Γ which is a model of φ. By the assumption T is also a model of ψ. Then it follows from Lemma 3 that T is a model of $u(\psi)$. As $u(\psi)$ contains no symbol from Γ and T is an extension of S on Γ, S is a model of $u(\psi)$.

Lemma 5. *Let φ be a formula over Σ, M an extended Herbrand structure over Σ and Var, and u an Γ-Unskolemisation for φ. Then*

$$M \models \varphi \quad \textit{implies that} \quad u(M) \models u(\varphi)$$

Proof. Assume that

$$M \models \varphi$$

Let $\forall \bar{x}(\psi) = \varphi$. Then for all sequences of terms $\bar{t}$ it holds that

$$M \models \psi(\bar{t})$$

As M is an extended Herbrand structure, it interprets every constant c by c. Therefore it follows from the substitution Lemma that

$$M \models (\psi\{\bar{c} \leftarrow \bar{y}\})(\bar{t},\ \bar{c})$$

Observe that $u(M)$ be the extended Herbrand structure obtained from M by renaming the domain elements $\bar{c}$ by $\bar{y}$. Thus

$$u(M) \models (\psi\{\bar{c} \leftarrow \bar{y}\})(\bar{t})$$

Finally by the definition of entailment and Unskolemisation it holds that

$$u(M) \models u(\varphi).$$

The following Lemma is used in both directions of the proof of Proposition A.

Lemma 6. *Let P be a logic program and a a ground atom. Then $P \models a$ iff $M_P \models a$.*

Proof. The direction from left to right is true since M_P is a model of P. For the other direction observe that M_P is equal to the least fixed point of the immediate consequence operator of P. An induction on the number of iterations of the immediate consequence operator shows that if a is in the least fixed point of T_P then a is entailed by P.

We now show Lemma 1.

Proof. (of Lemma 1) Let P be an RDFLog program over alphabet Σ. We need to show that $u(M_{s(P)}) \models P \quad t$ and $P \models u\left(\varphi_{M_{S(P)}}\right)$.

To show that $A_P \models P$, observe that by definition $M_{s(P)}$ is a model of $s(P)$. It therefore follows from Lemma 5 that $u(M_{s(P)})$ is a model of P.

For the second conjunct observe that as $s(P)$ is a logic program it follows from Lemma 6 that $s(P)$ entails each atom that is true in $M_{s(P)}$. Thus $s(P)$ also entails the canonical formula $\varphi_{M_{s(P)}}$ of $M_{s(P)}$. Let u be the inverse of s. As $s(P)$ is a finite set of finite formulas and φ is a ground formula it follows from Lemma 4 that $u \circ s(P) = P$ entails $u\left(\varphi_{M_{s(P)}}\right)$.

We now turn to the proof of Theorem 1. The following Lemma is used to show that every RDF graph which is entailed by an RDFLog program is also entailed by the operational semantics.

Lemma 7. *Let P be a logic program over alphabet Σ and M_P its minimal Herbrand model. Let g be an RDF graph and $\varphi^{\mathsf{g}} = \exists \bar{x}\,(\bigwedge \Phi)$ its canonical formula. Then the following statements are equivalent*

(a) $P \models \bigwedge \Phi\{\bar{x} \leftarrow \bar{t}\}$ *for some sequence of variables $\bar{x}$ and ground terms $\bar{t}$*
(b) $P \models \varphi^{\mathsf{g}}$
(c) $M_P \models \varphi^{\mathsf{g}}$

Proof. It is trivial that (a) implies (b). To see that (b) implies (c) observe that M_P is a model of P. To show that (c) implies (a) assume that (c) is true. As M_P is a Herbrand model there is a sequence of terms $\bar{t}$ such that $M_P, \bar{t}$ is a model of $\bigwedge \Phi(\bar{x})$. In addition, M_P interprets all terms by themselves. Thus it follows from the substitution lemma that M_P is a model of $\bigwedge \Phi\{\bar{x} \leftarrow \bar{t}\}$. Therefore M_P is a model of $a\{\bar{x} \leftarrow \bar{t}\}$ for every $a \in \Phi$. As $a\{\bar{x} \leftarrow \bar{t}\}$ is a ground atom it follows from 6 that $P \models a\{\bar{x} \leftarrow \bar{t}\}$. As this is true for every $a \in \Phi$ it holds that $P \models \bigwedge \Phi\{\bar{x} \leftarrow \bar{t}\}$ for some sequence of terms $\bar{t}$.

We finally prove Theorem 1

Proof. (of Theorem 1) Let P be an RDFLog program over Σ^{V} and $\varphi_P = u(\varphi_{M_{s(P)}})$. We first show that that for any RDF graph g over V

$$P \models \varphi_{\mathsf{g}} \quad \text{iff} \quad \varphi_P \models \varphi_{\mathsf{g}}$$

where φ_{g} is the canonical formula of g. The direction from right to left follows from the second conjunct of Theorem 1. For the direction from left to right let $\varphi_{\mathsf{g}} = \exists \bar{x}\,(\bigwedge \Phi)$ where Φ is a set of atoms over $\Sigma^{\mathsf{V}} \cup \mathit{Var}^{\mathsf{V}}$. Assume that $P \models \varphi_{\mathsf{g}}$. By Lemma 3 $s(P) \models \varphi_{\mathsf{g}}$ for any Skolemisation s of P. As $s(P)$ is a logic program it follows from Lemma 7 that there is a sequence $\bar{t}$ of terms such that $s(P) \models \bigwedge \Phi\{\bar{x} \leftarrow \bar{t}\}$. Thus for all $a \in \Phi\{\bar{x} \leftarrow \bar{t}\}$ it holds that $s(P) \models a$. As $M_{s(P)}$ is a model of $s(P)$ it follows that $M_{s(P)}$ is a model of a. Let $\bigwedge M_{s(P)}$ be the conjunction of all ground atoms which are true in $M_{s(P)}$. Then a is a conjunct in $M_{s(P)}$ and thus $\bigwedge M_{s(P)} \models a$. As this is true for any $a \in \Phi\{\bar{x} \leftarrow \bar{t}\}$ it holds that $\bigwedge M_{s(P)} \models \Phi\{\bar{x} \leftarrow \bar{t}\}$.

Thus $\bigwedge M_{s(P)} \models g$ there is a homomorphism μ from $M_{s(P)}$ to G. Observe that there is a mapping ν from $D^{M_{s(P)}}$ to $D^{u(M_{s(P)})}$ such that (i) $\nu(c^{M_{s(P)}}) = c^{u(M_{s(P)})}$ if $c \in \Sigma$, (ii) if $f \in \Gamma$ then $\nu(f(\bar{t})) = x_{f(\bar{t})}$ where $x_{f(\bar{t})}$ is a non-constant domain element in $D^{u(M_{s(P)})}$, and (iii) $R^{M_{s(P)}}(\bar{d})$ iff $R^{u(M_{s(P)})}(\nu(\bar{t}))$ for every relation symbol R in Σ. Observe that $\mu \circ \nu$ is a homomorphism from G to $u(M_{s(P)})$. Thus the operational semantics $u(M_{s(P)})$ of P entails φ^{g}.

It remains to show that $\varphi \models \varphi_{\mathsf{g}}$ iff $\Pi(\varphi) \models \varphi_{\mathsf{g}}$ if φ is a formula as in the definition of the normalisation operator and φ_{g} is the canonical formula of an RDF graph. The direction from right to left is immediate since $\Pi(\varphi) \models \varphi$. The other direction follows from the definition of Π and the special structure of RDF triples.