RDF Surfaces as a First-Order language for the Semantic Web

Dörthe Arndt[1], Jos De Roo[2], Patrick Hochstenbach[2,3], Rebekka Martens[1], Femke Ongenae[2], and Mathijs van Noort[2]

[1] Computational Logic Group, Technische Universität Dresden, Germany,
[2] IDLab, Ghent University - Imec, Belgium,
[3] Ghent University Library, Ghent University, Belgium,

Abstract. Inspired by the idea of RDF Redux, an RDF extension suggested by Pat Hayes, RDF Surfaces were recently developed and further specified by a W3C community group. The idea of RDF Surfaces is to add negation and explicit existential quantification to RDF and thereby gain the expressivity of first-order logic. RDF Surfaces come with a syntax and even with first implementations, but the semantics has so far only been defined informally. In this paper we aim to close this gap: we map RDF Surface graphs to first-order logic formulae and thereby define their semantics. We show that, restricted to RDF graphs, this semantics preserves simple entailment. That is, each RDF graph which entails another in it's first-order translation, also entails this graph according to RDF's simple entailment and vice versa. To test whether this semantics fully meets the informal specification, we furthermore provide `rs2fol`, an implementation which follows our mapping and translates RDF Surfaces in N3-based syntax to first-order logic in TPTP syntax. We apply this implementation on the various examples collected on the Web page of the RDF Surfaces reasoner EYE, run them with the theorem prover Vampire and compare the results with those of EYE. With the exception of a different understanding of lists – EYE treats these as first-class citizens – results of both approaches coincide. We thus provides a tool for entailment checking which is conform to the current specification. This tool will help future developers of RDF Surfaces reasoners to test their derivations for correctness and the community as a whole to better understand the logic and – if needed – to refine or even restrict it.

Keywords: RDF Surfaces · FOL · Reasoning · RDF · Semantics.

1 Introduction

RDF [6] can be seen as the most basic, but also most important, logic of the Semantic Web. The majority of other logical Web frameworks – for example OWL with its different profiles [3], Notation3 Logic (N3) [30], and SPARQL[11] – are based on (parts of) RDF. Most of these frameworks are compatible with RDF on a syntactic level. However, this does not hold for the semantics. For instance, OWL's description logic based direct semantics [20] and its RDF-based semantics [21], which directly extends RDF semantics, are not fully compatible. And while there is the possibility to include entailment

regimes in SPARQL, it mainly connects to RDF through subgraph matching. These observations motivated Pat Hayes to re-think RDF. In his ISWC 2009 keynote talk [12] he proposed RDF Redux as an improved version of RDF. RDF Redux is a direct extension of RDF, which adds explicit existential quantification and negation to RDF, with simple interpretations. As RDF already supports the conjunction of atomic formulae, it becomes as expressive as first-order logic (FOL) by these additions and thus also covers, for example, $\mathcal{SHOIN}$ [1], the formal base of OWL DL. Following this idea, an W3C community group was formed[4] and RDF Surfaces were developed [15,17]. A Web logic with a concrete realisation in an N3-based syntax. Reasoning on this framework is supported by the EYE reasoner [7], which also hosts many examples discussed in the community group on its Web page [8], by Latar [14] and by Tension [27]. These implementations rely on an informal specification of the logic. The formal semantics of RDF Surfaces has not been defined yet.

In this paper we fill this gap. Following the informal descriptions and specifications detailed by Hochstenbach et al. [17], we define the abstract syntax of RDF Surfaces and map the formulae we retrieve directly to first-order logic. Our work builds upon the work of De Bruijn and Heymans who provided a mapping from RDF to first-order logic that preserves simple entailment [4]. Just as De Bruijn and Heymans, we understand graphs as conjunctions of triple statements where the blank nodes are assumed to be existentially quantified. We furthermore include negation and support for explicit existential quantification. With this mapping, we provide FOL-based semantics for RDF Surfaces. To test whether our mapping is in line with the informal semantics of the logic, we furthermore implemented `rs2fol`, a Kotlin program, which employs our mapping to translate concrete N3 representations of RDF Surfaces to TPTP syntax [25] – a standard format for first-order theorem provers – and performs entailment checking using the prover Vampire [19]. In our evaluation, we use this program to reason over the examples collected on the Web page of EYE [8]. These reflect the common expectations on the logic as they mainly contain use cases discussed in the community group and tests sent in by users. We compare our reasoning results to those of EYE, the most advanced among the different `rs2fol` reasoners. In a second round, we use our own test cases originated from FOL with `rs2fol` and EYE to exemplify how the development of RDF surfaces and its reasoners can benefit from `rs2fol`.

The remainder of the paper is structured as follows: In Section 2, we explain the idea of RDF Surfaces in more detail and provide an example. After that, in Section 3, we introduce RDF Surfaces in a more formal way. In Section 4, we then define our mapping from RDF Surfaces to first-order logic which we implemented in our tool `rs2fol` which we describe in Section 5. This is followed, in Section 6, by the practical evaluation of our work. In Section 7, we discuss related work and end our paper with a conclusion (Section 8).

2 Idea and Examples

Before formally introducing the necessary concepts, we provide an informal introduction to RDF Surfaces to make the reader familiar with the general idea. As RDF Sur-

[4] https://www.w3.org/community/rdfsurfaces/

faces extends RDF with simple interpretations, we start by providing a triple:[5]

```
:socrates a :Human .                                    (1)
```

Informally speaking, this triple can be interpreted as *"Socrates is a human."*, or, in FOL (making use of the predicate tr to represent triples):[6]

$$tr(socrates, type, Human) \tag{2}$$

Applying RDF's simple entailment [13], we can replace the constant `:socrates` by a blank node:

```
_:x a :Human .                                          (3)
```

which informally means *"There exists a human."*, or in FOL:

$$\exists x.tr(x, type, Human) \tag{4}$$

With RDF Surfaces, we make this implicit quantification explicit. Instead of Triple 3, we thus write:

```
(_:x) log:onPositiveSurface { _:x a :Human . } .        (5)
```

We furthermore have the possibility to state negative information. If we write

```
(_:x) log:onNegativeSurface { _:x a :Human . } .        (6)
```

this means that no humans exist, or in FOL[7]:

$$\neg\exists x.tr(x, type, human) \tag{7}$$

As this construct adds negation and explicit quantification, we can now express first-order operators. To state universal quantification, we use nested negation. The equivalence of

$$\forall x.tr(x, likes, cheese) \equiv \neg(\exists x.\neg tr(x, likes, cheese))$$

expressing the fact that *"everyone likes cheese"*, can thus be expressed in RDF Surfaces as follows:

```
(_:x) log:onNegativeSurface {                           (8)
     () log:onNegativeSurface { _:x :likes :cheese . } . } .
```

[5] To save space, we omit the prefix declarations in this paper.

[6] An alternative representation would be $type(socrates, human)$. We use the triple notation following [4] because RDF Surfaces allow variables in predicate position, more on that in Section 4.

[7] Note that this construct makes the previous one only syntactic sugar. Triple 5 could also be written as `() log:onNegativeSurface {(_:x) log:onNegativeSurface { _:x a :Human.}.}`. We include positive surfaces to improve readability.

Similarly, we can express disjunction. The statement *"socrates likes cheese or wine"* can be written in FOL as follows:

$$\begin{aligned} & tr(socrates, likes, cheese) \vee tr(socrates, likes, wine) \\ \equiv & \neg(\neg tr(socrates, likes, cheese) \wedge \neg tr(socrates, likes, wine)) \end{aligned} \tag{9}$$

This leads to the RDF Surface:

```
() log:onNegativeSurface {                                    (10)
     () log:onNegativeSurface { :socrates :likes :cheese . } .
     () log:onNegativeSurface { :socrates :likes :wine . } .
} .
```

The idea is, thus, that by the simple addition of negation and explicit existential quantification to RDF, we are able to express first-order formulae in a Semantic Web format. As a last example to illustrate the idea, we translate the rule that *"all humans are mortal"*, that is

$$\begin{aligned} & \forall x.tr(x, type, human) \rightarrow tr(x, type, mortal) \\ \equiv & \neg\exists x.(tr(x, type, human) \wedge \neg tr(x, type, mortal)) \end{aligned} \tag{11}$$

to RDF Surfaces:

```
(_:x) log:onNegativeSurface {                                 (12)
     _:x a :Human .
    () log:onNegativeSurface {
        _:x a :Mortal .
    } .
} .
```

RDF Surfaces thus also cover classical rules as for example present in N3 or SWRL [18]. More complicated constructs like disjunction in the conclusion of rules allow us to cope, for example, with the union operator of OWL DL [29]. RDF Surfaces provides a framework to express FOL in a Semantic Web format.

3 RDF Surfaces

After our informal introduction of RDF Surfaces and the idea behind our mapping, we next introduce the logic in a more formal way. Our definitions are inspired by Hochstenbach et al. [17]. The abstract syntax of RDF Surfaces is an extension of RDF's abstract syntax, more precisely, of generalised RDF [5]:

Definition 1. *The set T of* RDF terms *consists of a set B of blank nodes, a set L of literals, and a set* IRI *of IRIs. We call a triple $t \in T \times T \times T$ a* generalised RDF triple. *A set of generalised RDF triples is called a* generalised RDF graph.

The reason to use generalised RDF instead of the regular recommendation is mainly that we want to avoid that the consequences of our reasoning cannot be expressed – rules could produce triples with literals in subject position – and that the logic should be strong enough to express frameworks like N3 or SPARQL where variables in predicate position are allowed.[8] Generalised RDF triples can now form part of a Hayes triple:

Definition 2. *A* Hayes triple *(H-triple) is a triple of the form $\langle Gr, s, H\rangle$ where Gr is a (possibly empty) list of blank nodes, called graffiti; $s \in \{\mathrm{nS}, \mathrm{pS}\}$ is the surface type; and H is a (possibly empty) graph, that is a set of generalised RDF triples and H-triples. We call H a* Hayes graph *(H-graph).*

Connecting our abstract syntax to the concrete example provided in Formula 5, we note that there, the list `(_:x)` is the grafitti, pS (for positive surface) is the surface type, and the graph containing Triple 3 is the H-graph. Note that with the introduction of explicit quantification, blank nodes can now represent free variables. From an RDF Surfaces point of view, the blank node `_:x` occurs as a free variable in Triple 3. We introduce the concept of free variables in a more formal way:

Definition 3. *Let H be an H-graph, $t = \langle t_1, t_2, t_3\rangle$ a generalised triple, and $h = \langle Gr, s, H\rangle$ an H-triple. We recursively define the set* fr *of free variables occurring in H, t and h as follows:*

$$\begin{aligned} \mathrm{fr}(t) &= \{t_1, t_2, t_3\} \cap B \\ \mathrm{fr}(h) &= \mathrm{fr}(H) \setminus \{x | x \textit{ occurs in } Gr\} \\ \mathrm{fr}(H) &= \bigcup_{x \in G} \mathrm{fr}(x) \end{aligned}$$

For technical reasons – RDF Surfaces aims to support plain RDF – RDF Surfaces semantics assumes the existential closure when interpreting H-graphs containing free variables. That is, Triple 3 is understood as Formula 4 above and the following surface (which is a slight variation of Formula 6)

```
() log:onNegativeSurface {_:x a :Human.}.
```
(13)

means

$$\exists x. \neg tr(x, type, human) \tag{14}$$

or in words *"there exists something which is not a human"*. Having introduced the syntax of RDF Surfaces, we provide the mapping to first-order logic in the next section.

4 Mapping RDF Surfaces to First-Order Logic

For our definitions, we briefly recap first-order logic and its semantics. For a more detailed discussion, we refer to the standard textbooks (e.g., Ebbinghaus et al. and Enderton [9,10]).

[8] Note, however, that due to SPARQL's non-monotonicity and built-in functions, RDF Surfaces will most likely only be suitable to cover parts of it.

Definition 4. *Given the disjoint sets V of variables, F of function symbols, each with a fixed arity $n \geq 1$, R of relation symbols, each with a fixed arity $n \geq 1$, and C of constants. We define the set $\mathscr{T}$ of FOL terms as follows: each $v \in V$ is a term, each $c \in C$ is a term, if $t_1, \ldots, t_n \in \mathscr{T}$ and $f \in F$ has arity n, then $f(t_1, \ldots, t_n)$ is a term. The set $\mathscr{F}$ of first-order formulae is then defined as: if $t_1, \ldots, t_n \in \mathscr{T}$ and $r \in R$ has arity n, then $r(t_1, \ldots, t_n) \in \mathscr{F}$, if $\phi, \psi \in \mathscr{F}$ then $(\phi \wedge \psi) \in \mathscr{F}$, if $\phi \in \mathscr{F}$, then $\neg\phi \in \mathscr{F}$, and if $v \in V$ and $f \in \mathscr{F}$ then $\exists x.f \in \mathscr{F}$*

Note that our definition omits the connectives $\vee, \rightarrow$ and $\leftrightarrow$ as these can be seen as syntactic sugar – $\phi \vee \psi$ can, for example, be expressed as $\neg(\neg\phi \wedge \neg\psi)$. We chose this definition to emphasize the strong connection between FOL and RDF Surfaces. For the same reason, we also omit the universal quantifier $\forall$: $\forall x.\phi$ can be expressed as $\neg\exists x.\neg\phi$.

For this syntax, we now also briefly recap the semantics:

Definition 5. *A structure $\mathfrak{A}$ for the sets F, R and C consists of a non-empty set $\mathscr{A}$, the domain, and of a mapping $\mathfrak{a}$ such that $\mathfrak{a}(c) \in \mathscr{A}$ for all $c \in C$, $\mathfrak{a}(r) \in 2^{\mathscr{A}^n}$ for all $r \in R$ with arity n, and $\mathfrak{a}(f)$ maps to an n-ary function in $\mathscr{A}$ if $f \in F$ and f has arity n. An assignment is a mapping $\beta : V \rightarrow \mathscr{A}$, for $a \in \mathscr{A}$ and $x \in V$, we write $\beta\frac{a}{x}$ to indicate a mapping which behaves as β with the exception that it maps x to a. We call a pair of $\mathfrak{I}(\mathfrak{A}, \beta)$ an interpretation and define $x^{\mathfrak{I}}$ as follows: $c^{\mathfrak{I}} = \mathfrak{a}(c)$ for $c \in C$, $x^{\mathfrak{I}} = \beta(x)$ for $x \in V$, $(f(t_1, \ldots, t_n))^{\mathfrak{I}} = (\mathfrak{a}(f))(t_1^{\mathfrak{I}}, \ldots, t_n^{\mathfrak{I}})$. Let ϕ and ψ be formulae. We recursivly define: $\mathfrak{I} \models r(t_1, \ldots, t_n)$ iff $\langle t_1^{\mathfrak{I}}, \ldots, t_n^{\mathfrak{I}} \rangle \in \mathfrak{a}(r)$, $\mathfrak{I} \models \phi \wedge \psi$ iff $\mathfrak{I} \models \phi$, $\mathfrak{I} \models \psi$, and $\mathfrak{I} \models \neg\phi$ iff $\mathfrak{I} \not\models \phi$, and $\mathfrak{I} \models \exists x\phi$ iff there exists an $a \in \mathscr{A}$ such that $\mathfrak{I}\frac{a}{x} \models \phi$. We call a formula ϕ satisfiable if there exists an interpretation $\mathfrak{I}$ such that $\mathfrak{I} \models \phi$, in this case we call $\mathfrak{I}$ a* model *of ϕ. We say that a formula ϕ entails a formula ψ, written as $\phi \models \psi$, if for each interpretation $\mathfrak{I}$ with $\mathfrak{I} \models \phi$ also $\mathfrak{I} \models \psi$ holds.*

As RDF does not support functions, these will also not play a role in our translation presented in this section. However, they will be used in our evaluation (Section 6.2) when we talk about lists. For this reason, they are included here. We now come to the definition of our mapping, which takes a formula in RDF Surfaces and translates it to FOL just as indicated by the examples in Section 2. This mapping relies on a term mapping $\ell : T \rightarrow V \cup C$, where each blank node is mapped to a variable, and each IRI and each literal is mapped to a constant. For simplicity, we assume this mapping to be injective. Note, however, that the mapping can be modified for literals such that these map to their canonical representation. This would allow us to handle datatype reasoning.

Definition 6. *Given an H-Graph G and an injective term mapping $\ell : T \rightarrow V \cup C$. We recursively define the translation mapping m from H-graphs to FOL as follows:*

- *for each generalised RDF triple $\langle s, p, o \rangle$: $m(\langle s, p, o \rangle) = tr(\ell(s), \ell(p), \ell(o))$;*
- *for each positive H-triple $\langle (b_1, \ldots b_n), \mathrm{pS}, H \rangle$:*
  $m(\langle (b_1, \ldots b_n), \mathrm{pS}, H \rangle) = \exists \ell(b_1) \ldots \exists \ell(b_n) m(H)$;
- *for each negative H-triple $\langle (b_1, \ldots b_n), \mathrm{nS}, H \rangle$:*
  $m(\langle (b_1, \ldots b_n), \mathrm{nS}, H \rangle) = \forall \ell(b_1) \ldots \forall \ell(b_n) \neg(m(H))$;
- *for each H-graph H: $m(H) = \bigwedge_{t \in H} m(t)$*

Note that the translation function can produce FOL formulae containing free variables. We interpret these formulae as existentially closed, that is, given an H-graph H with a set of free variables $\mathrm{fr}(H) = \{x_1, \ldots, x_n\}$, we translate H via m and add quantifiers, such that $\exists \ell(x_1) \ldots \exists \ell(x_n) m(H)$. As a shortcut for the existential closure we will from now on write $\exists \mathbf{x}.m(H)$.

Our mapping relies on a term mapping ℓ. Note that the concrete choice of ℓ is not relevant for our purposes as long as ℓ is fixed and injective. We therefore omit ℓ from now on and assume it to be arbitrary but fixed and injective. We now use our mapping m to define the semantics of RDF Surfaces:

Definition 7. *Let H and G be H-graphs. We say that H is* h-satifiable*, if there exists a first-order interpretation $\mathfrak{I}$ such that $\mathfrak{I} \models \exists \mathbf{x}.m(H)$. We say, that H h-entails G, written as $H \models_h G$ if for each first-order interpretation $\mathfrak{I}$, it holds that $\mathfrak{I} \models \exists \mathbf{x}.m(G)$ if $\mathfrak{I} \models \exists \mathbf{x}.m(H)$. If $\mathfrak{I} \models \phi$, we call $\mathfrak{I}$ a* model *of ϕ.*

Before evaluating the accuracy of our mapping with the existing examples for RDF Surfaces reasoning, we would like to briefly discuss how h-entailment behaves for generalised RDF graphs, that is, for graphs not containing H-triples. Recall that the semantics of these graphs is defined as follows [13]:

Definition 8. *A simple interpretation I is a structure consisting of a non-empty set* IR *of resources, a set* IP*, called the set of properties, a mapping* $\mathrm{IEXT} : \mathrm{IP} \to 2^{\mathrm{IR} \times \mathrm{IR}}$*, a mapping* $\mathrm{IS} : \mathrm{IRI} \to (\mathrm{IR} \cup \mathrm{IP})$*, a partial mapping* $\mathrm{IL} : L \to \mathrm{IR}$*. For a ground graph (a graph without blank nodes) G, we recursively apply I as follows: If $l \in L$: $I(l) = \mathrm{IL}(l)$, for $c \in$ IRI: $I(c) = \mathrm{IS}(c)$, if $\langle s, p, o \rangle$ a ground triple: $I(\langle s, p, o \rangle) =$* true *iff $\langle I(s), I(o) \rangle \in \mathrm{IEXT}(I(p))$, and $I(G) =$* true *iff $I(t) =$* true *for all triples $t \in G$. For a mapping $A : B \to \mathrm{IR} \cup \mathrm{IP}$, we define $[I+A](x) = A(x)$, if $x \in B$, $[I+A](x) = I(x)$ else. If G is a generalised RDF graph then $I(G) =$* true *iff there exists a mapping A such that $[I+A](G) =$* true*. We call G simply satisfiable if there exists an interpretation such that $I(G) =$* true*. We say that a graph G simply entails a graph F, written as $G \models_s F$, if for all interpretations I with $I(G) =$* true*, we also have $I(F) =$* true*.*

If we apply the mapping specified in Definition 6 on only generalised RDF graphs, we get a conjunction of triples in which the blank nodes are existentially quantified. The graph

```
:socrates :knows _:x. _:x :name "Plato".
```
(15)

becomes

$$\exists x(tr(socrates, knows, x) \wedge tr(x, name, Plato)) \tag{16}$$

For these basic graphs, we can directly transform simple RDF interpretations to first-order interpretations and vice versa while preserving the truth:

Lemma 1. *Let G be a generalised RDF graph, then, the following holds:*

$$G \text{ is simply satisfiable} \quad \textit{iff} \quad G \text{ is h-satisfiable.}$$

Proof. If $G = \{\langle s_i, p_i, o_i \rangle | 1 \leq i \leq m\}$ is a generalised RDF graph, $\exists \mathbf{x}.m(G)$ is of the form

$$g = \exists \ell(x_1) \ldots \exists \ell(x_n) \bigwedge_{1 \leq i \leq m} tr(\ell(s_i), \ell(p_i), \ell(o_i))$$

Note that this FOL formula contains one single predicate symbol, namely tr. We show both directions separately.

"$\Leftarrow$": Let $\mathfrak{I}$ be an FOL interpretation of g such that $\mathfrak{I} \models g$. We construct a simple interpretation $I = (\mathrm{IR}, \mathrm{IP}, \mathrm{IEXT}, \mathrm{IS}, \mathrm{IL})$ for G as follows: $\mathrm{IR} := \mathscr{A}$, $\mathrm{IP} := \{p \in \mathscr{A} \,|\, \exists s, o \in \mathscr{A} : \langle s, p, o\rangle \in \mathfrak{a}(tr)\}$, $\mathrm{IEXT}(p) := \{\langle s, o\rangle | \langle s, p, o\rangle \in \mathfrak{a}(tr)\}$, $\mathrm{IS}(c) := \mathfrak{a}(\ell(c))$ if c is an IRI and $\mathrm{IL}(l) := \mathfrak{a}(\ell(l))$, if l is a literal. To see, that $I(G) = \text{true}$, we need to take a closer look at the blank nodes occuring in G. As $\mathfrak{I} \models \exists \ell(x_1) \ldots \exists \ell(x_n) m(G)$, there exist $a_1, \ldots, a_n \in \mathscr{A}$ such that $\mathfrak{I}\frac{a_1}{\ell(x_1)} \ldots \frac{a_n}{\ell(x_n)} \models m(G)$. We define A such that $A(x_i) := a_i$, for $1 \leq i \leq n$. With that A we get $[I+A](G) = \text{true}$.

"$\Rightarrow$": Let $I = (\mathrm{IR}, \mathrm{IP}, \mathrm{IEXT}, \mathrm{IS}, \mathrm{IL})$ such that $I(G) = \text{true}$. To construct a model $\mathfrak{I}$ for g, we define: $\mathscr{A} := \mathrm{IR} \cup \mathrm{IP}$, $\mathfrak{a}(t) := I(\ell^{-1}(t))$ if t is a constant. Note that this mapping is well-defined because ℓ is injective and because if $I(G) = \text{true}$, then $\mathrm{IL}(l)$ is defined for all literals occurring in G. For tr we set $\mathfrak{a}(tr) := \{\langle s, p, o\rangle | \exists p \in \mathrm{IP} : \langle s, o\rangle \in \mathrm{IEXT}(p)\}$. As $I(G) = \text{true}$, there exists furthermore a mapping $A : B \to \mathrm{IR} \cup \mathrm{IP}$, such that $[I+A](G) = \text{true}$. We use the values of that mapping to obtain $\mathfrak{I}\frac{A(x_1)}{\ell(x_1)} \ldots \frac{A(x_n)}{\ell(x_n)} \models m(G)$. We thus get $\mathfrak{I} \models \exists \ell(x_1) \ldots \exists \ell(x_n) m(G)$. □

As a direct consequence of that lemma, we get the theorem below, which means that if we limit h-entailment to RDF graphs, it behaves just as simple entailment:

Theorem 1. *For two generalised RDF graphs G and F the following holds:*

$$G \models_s F \quad \textit{iff} \quad G \models_h F$$

Proof. If $G \models_s F$ and $\mathfrak{I} \models \exists \mathbf{x}.m(G)$. Then, according to Lemma 1, we can construct a simple interpretation I for G such that $I(G) = \text{true}$. According to our assumption, we then also find a simple interpretation I' such that $I'(F) = \text{true}$. we again apply Lemma 1 and obtain a model $\mathfrak{I}' \models \exists \mathbf{x}.m(F)$. The other direction follows with the same argumentation. □

5 Implementation

In the previous section, we introduced a mapping from RDF Surfaces to FOL and used this to define their semantics. In addition to the fact that this helps us to better understand this new Web logic, we also have a practical advantage: FOL is supported by a wide range of tools performing reasoning. To make use of this tooling, but also to further test our mapping, we developed `rs2fol`, a Kotlin program which translates RDF surfaces to first-order logic and then uses a first-order theorem prover to perform entailment checking. In this section, we explain our implementation in detail. All code is available at https://github.com/RebekkaMa/rs2fol.

`rs2fol` directly implements the translation function m introduced in Definition 6 and converts RDF Surfaces, expressed in N3 syntax [17] – the syntax used in our motivational examples (Section 2) – into first-order logic formulae, written TPTP format [25]. TPTP is a very common input format for first-order theorem provers, and we chose it to be able to connect to different tools. In TPTP, facts are marked as *axioms*. For example, Formula 1 (*"Socrates is a human."*) is represented as:

```
fof(formula1,axiom,triple('socrates','type','Human')).        (17)
```

To test for whether or not a formula is a logical consequence of the others, potential consequences are marked as *conjecture*. If we, for example, want to know whether Formula 3 (*"There exists a human."*) is a logical consequence of the previous one, we add

```
fof(formula3,conjecture,
        ? [X] : (triple(X,'type','Human'))).
```
(18)

The question mark in this example indicates an existential quantifier.

`rs2fol` takes these translations and hands them over to a theorem prover. By using TPTP as a representation format for FOL, we made our implementation compatible with different reasoning tools. Currently, `rs2fol` employs Vampire [19], which we have chosen for its persistently convincing results in prover competitions [26] and its good technical support. With Vampire we have two options. We either test whether one or more of the conjectures follow from the axioms (entailment checking) or, of no conjecture is given, it tests for satisfiability of the axioms. Employing Definition 7, that means that our implementation `rs2fol` performs h-entailment and h-satisfiability checking. Assuming that the calculus of Vampire is sound and complete and that our definition of the mapping actually meets the expectations informally stated about RDF surfaces, we provided the first sound and complete entailment checker for h-entailment.[9]

6 Evaluation

In the previous section, we introduced `rs2fol`, our tool to perform h-entailment checking according to our definition. We already discussed that h-entailment truly extends RDF's simple entailment. As a next step, we would like to use `rs2fol` to evaluate whether our definition is in line with the informal specification. To do so, we perform two tests: (1) we check whether our proposed transformation from RDF Surfaces to first-order logic is correctly implemented, (2) being sure that `rs2fol` works as expected, we evaluate whether `rs2fol` supports the test cases for RDF Surfaces which exist so far. For (1), we considered a calculus, adopted from [9], and created test in the N3-based syntax reflecting the different derivation rules. For (2) we compared the reasoning results of the EYE reasoner to those of `rs2fol` on the example cases provided on EYE's Web page [8]. Below, we discuss the two data sets, the tests performed, and the respective test results separately. Afterwards we discuss tests we created motivated by these results.

6.1 Validity of the implementation

From the point of view of first-order logic, RDF Surfaces is equipped with conjunction (two statements on a single surface), negation (the negative surface type), and existential quantification (blank node graffiti on a negative surface). We consider a first-order logic syntax with operators $\wedge$, $\neg$ and $\exists$. Via equivalences $\phi \wedge \psi \equiv \neg(\neg\phi \wedge \neg\psi)$, $\phi \rightarrow \psi \equiv \neg(\phi \wedge \neg\psi)$ and $\forall \mathbf{x}.\phi \equiv \neg(\exists \mathbf{x}.\neg\phi)$, it follows that said logic has full first-order expressivity.

[9] Note, however, that FOL entailment- and satisfiability checking is known to be undecidable. That is a limitation which is unavoidable for RDF Surfaces given its complexity.

`rs2fol` translates h-graphs to first-order formulae. The theoretical validity of this transformation follows from Theorem 1. In order to additionally verify the validity of our implementation itself, we verify whether the output of `rs2fol` behaves as intended; we consider a set of calculus rules for FOL, taken from Ebbinghaus et al. [9], but adapted to accommodate for the syntactical difference of $\wedge$ instead of $\vee$:

$$\frac{\Gamma\ \phi}{\Gamma'\ \phi}\ \text{for}\ \Gamma \subseteq \Gamma', \qquad \frac{}{\Gamma\ \phi}\ \text{for}\ \phi \in \Gamma, \qquad \frac{\Gamma\ \psi\,\phi, \Gamma\ \neg\psi\,\phi}{\Gamma\ \phi}, \tag{19a}$$

$$\frac{\Gamma\ \neg\phi\ \psi, \Gamma\ \neg\phi\ \neg\psi}{\Gamma\ \phi}, \qquad \frac{\Gamma\ \phi, \Gamma\ \psi}{\Gamma\ (\phi \wedge \psi)}, \qquad \frac{\Gamma\ \phi\ \chi}{\Gamma\ (\phi \wedge \psi)\ \chi}, \tag{19b}$$

$$\frac{\Gamma\ \phi\frac{t}{x}}{\Gamma\ \exists x\,\phi}, \qquad \frac{\Gamma\ \phi\frac{y}{x}\ \psi}{\Gamma\ \exists x\phi\ \psi} \tag{19c}$$

Each of the rules in Equations 19a-19c of the calculus was encoded into RDF Surfaces. We translated the premisses and a negation of the consequent[10] and performed satisfiability checking with `rs2fol` to detect that the positive consequent is entailed by the succedent. This was the case for all our files. While this alone can not be seen as proof that `rs2fol`'s h-entailment checking is indeed complete and correct, we see this as a strong indication that the translation itself works. In terms of logical correctness and completeness, the implementation relies on Vampire and its underlying calculus.

6.2 Conformence to the informal semantics

After verifying that the code of `rs2fol` works as intended, we utilize `rs2fol` to answer the following question: Does our definition of h-entailment fully reflect the informal semantics? Of course, this question is difficult to answer because it lies in the nature of an informal specification with different contributers, that expectations on the logic differ. However, there are implementations available which indicate the implementer's intuition and there exist test cases. For our tests, we chose to work with the reasoner EYE [28], a Notation3 engine that also supports RDF Surfaces reasoning. We chose that reasoner because it is the most advanced among the current implementations. As test files, we chose the test folder [8] on EYE's web page which does not only contain the test cases invented by the developers of EYE, but also cases that were discussed in the W3C community group or sent in by users. In this sense, the test cases reflect how the informal semantics is supposed to work.

In our tests, we compared the reasoning results of the EYE reasoner with those of `rs2fol`. At the time of testing, the test folder contained 97 files,[11] among these, 43 files included special features like customized built-in predicate or unusual syntax that are specific to EYE and were therefore excluded. To the remaining 54 files we used for testing we added one modified example, `peano_short.n3s`, derived from one of the unsupported files by reducing and removing unsupported elements.[12] The reason

[10] The test files are available at https://github.com/RebekkaMa/rs2fol/tree/master/examples/sequent-calculus/minimal-example.

[11] A snapshot is available at https://github.com/eyereasoner/rdfsurfaces-tests/tree/94eb1b74c70b51f34f67b15d59e8e02ebaf4a96b

[12] Our tests folder is available at https://github.com/RebekkaMa/rs2fol/tree/master/examples/rdfsurfaces-tests.

for that addition was that we wanted to include a file containing RDF list structures and these were not present in other files. EYE does not only perform entailment checking, but also has a query function that produces logical consequences of the input. We tested whether the output produced by the reasoner is h-entailed by the formulae. Our tests use a Bash script that systematically processes each of the 55 RDF Surfaces files. For each file, the script initially invokes EYE (version 10.16.4) to obtain the reasoning results. Subsequently, it executes `rs2fol` (version 1.0.0) which converts the currently processed file into a TPTP axiom and EYE's reasoning result into a TPTP conjecture formula. These two formulae are then passed to Vampire (version 4.7) for actual consequence checking.

We executed this script and observed that `rs2fol` confirmed all entailments performed by EYE with the exception of `peano_s.n3s`, the file we added. Further investigation showed that the reason for the disagreement was indeed the presence of lists: RDF's list construct heavily depends on blank nodes. In RDF, the list notation in the RDF triple

```
(:a :b) a rdf:List .
```
(20)

is syntactic sugar for

```
_:b1 a rdf:List . _:b1 rdf:first :a; rdf:rest _:b2 .
     _:b2 rdf:first :b; rdf:rest rdf:nil .
```
(21)

In contrast to that, N3 understands lists as first-class citizens [2], and given that EYE is also an N3 reasoner, it follows this interpretation. For list interpretations supporting this interpretation, two lists having the same first and rest elements are interpreted equally, even if the lists themselves are represented by different blank nodes. In our test, two such lists where produced by different formulae and were thus equl in EYE but not for `rs2fol`.

It was rather easy to modify the translation `rs2fol` performs to make it understand lists the same way as EYE does. We added a binary first order function `list` and represented lists through nested functions, `(:a :b)` from above becomes `list(a,list(b,nil))`, and sucessfully repeated the test with the modification. However, RDF surfaces were introduced as a minimal extension or RDF with simple entailment [17] and this means that lists need to either be in line with the semantics of this framework or the difference should be reflected in the model. If we go for the first approach, it still needs to be clarified where exactly the blank nodes introduced by the use of `()`-lists like the one exemplified in Formula 20 need to be placed. Our original implementation assumes this to be on the next surrounding surface of the list, but further tests are needed. We expect similar issues to happen with RDF-star given that – at least at the time of writing – it also adds syntactic sugar relying on blank nodes.

6.3 Universal quantification

Motivated by the previous findings on lists, we performed a few more tests. Primary goal of these tests were to spot differences between `rs2fol` and EYE, as a seconary goal, we wanted to see how `rs2fol` entailment checking can help developers to spot

incompleteness in their reasoning results. We extended the examples using the sequent rules from Section 6.1 to get more complex examples[13]. We observed that EYE and `rs2fol` agree in the mayority of cases, but we could spot differences in examples where unresticted universal statements are derived through reasoning and then need to be applied. To illustrate this problem, we display a small example in Listing 1. Applying our mapping from Definition 4, the h-graph in the listing has the following FOL representation:

$$\begin{aligned} &tr(bob, is, laughing) \wedge \neg tr(sue, is, happy) \\ &\wedge \neg(\exists x \exists y\, tr(x, is, laughing) \wedge \neg tr(y, is, happy)) \end{aligned} \tag{22}$$

which is equivalent to

$$\begin{aligned} &tr(bob, is, laughing) \wedge \neg tr(sue, is, happy) \\ &\wedge \forall x \forall y\, tr(x, is, laughing) \rightarrow tr(y, is, happy) \end{aligned} \tag{23}$$

It is easy to see that this formulae do not have a model, or in other words, that $tr(sue, is, happy)$ is a logical consequence of $tr(bob, is, laughing)$ and $\forall x \forall y\, tr(x, is, laughing) \rightarrow tr(y, is, happy)$. Currently, the reasoning process in EYE is able to correctly derive $\forall x\, tr(y, is, happy)$, but it does not apply this universal fact to the instance *sue*. The reason seems to be in the algorithm it applies, which relies on materialization and needs to carefully select which facts it produces to avoid infinite productions. The reasoner itself provides special surfaces, to allow the user to guide the reasoning process. This observation leads to interesting discussions: while we defined semantics for h-entailment, we did not discuss how reasoning should work in practice and how and for which use cases it should be optimized. Do we want a logic with a control aspect similar to the idea of the cut operator in Prolog? Do we want to introduce fragments allowing performant reasoning? This discussion should be held in the community, but with our implementation, we provided a tool which will be very helpful in this regard as it can be used to spot shortcomings in implementations which should then be carefully inspected.

7 Related Work

Before concluding our paper, we would like to place our contribution into context. As mentioned in the beginning of this paper, the initial idea leading to the development of RDF Surfaces was given by Pat Hayes in a key note talk at ISWC 2009 [12]. In this talk he discussed RDF in the context of the Semantic Web and the shortcomings he encounters. He collected his requirements on a WeB LOGIC (BLOGIC) and made a suggestion, RDF Redux. This logic supports existential quantification, conjunction and negation just as RDF Surfaces. The suggestion of Pat Hayes was inspired by Existential graphs, in particular Beta graphs, which have been invented by Charles Sanders Peirce (1839-1914) [23] and further developed by John Sowa [24]. The idea behind this logical

[13] The test folder is available at: https://github.com/RebekkaMa/rs2fol/tree/master/examples/sequent-calculus/combined-rules

```
:bob :is :laughing .

() log:onNegativeSurface { :sue :is :happy . } .

(_:x _:y) log:onNegativeSurface {
        _:x :is :laughing .
        () log:onNegativeSurface { _:y :is :happy . } .
} .
```

Listing 1: Example: If someone is laughing, everybody is happy. Bob is laughing and sue is not happy. This should lead to a contradiction which can be detected by `rs2fol` but in this format not (yet) by EYE.

framework is to provide reasoning with a graphical calculus. An interesting aspect here is that graphs are put into contexts which, according to Hayes, also happens with data in the Web. And there, we can still see a connection tp RDF Surfaces which otherwise neglects the visual aspect. The mapping in this paper was inspired by the work of De Bruijn and Heymans who mapped plain RDF to F-logic [4] – an FOL based framework for knowledge representation with the goal to better understand its semantics. Similarly, Michael Schneider who is the editor of OWL's RDF-based semantics [21] provided in collaboration with Geoff Sutcliffe a mapping from that semantics (back then OWL-full) to TPTP [22], in order to perform reasoning. The definition of the full-profile as well as the mapping can be seen as an attempt to combine RDFS with DL. In that sense, RDF Surfaces provides an alternative approach which extends RDF simple entailment to FOL in order to get campatibility with description logic.

8 Conclusion

In this paper we mapped RDF surfaces to first-order logic and thereby provided a definition of its formal semantics. We evaluated this semantics in two ways: (1) we proved that it supports RDF simple entailment, that is, if two RDF graphs entail each other under simple entailment, they also h-entail each other and that if they h-entail each other they also simply entail each other. (2) We tested whether our definition is in-line with the informal semantics. To perform this test, we developed `rs2fol`, a tool which executes our maping and then uses the theorem prover Vampire to perform entailment checking. Here, we used `rs2fol` to verify the reasoning results of the reasoner EYE on different examples. Our tests showed that `rs2fol` and EYE agree in most cases, but we could also spot differences. One important point was that neither our definitions nor previous work [16,17,15] define how concepts which introduce blank nodes through syntactic sugar should handle these. EYE treats lists as first-class citizens. This makes them easier to process and that could also be an option for RDF surfaces, it would, however, means that RDF Surfaces would deviate from the original RDF. In this context, it will also be challenging to integrate RDF-star, once RDF 1.2. is released. Another point we observed was that the results of EYE differ from those of `rs2fol` in cases where unrestricted universally quantified facts are derived which need to be applied to data.

The reason for this finding was most likely that EYE is currently under development and does not yet fully support RDF Surfaces. It is, however, an open question whether it is really useful to fully support the whole logic with a reasoner as this comes with a price: first-order reasoning is not decidable and even for decidable cases often very slow. Given its expressivity, RDF Surfaces will have the same problem, so it makes sense to optimize for the most common use cases or to provide other measures to guide the reasoning. EYE does this by having extra surface types. Such things as well as possible profiles emposing restrictions need to be discussed in the community.

Apart from the obvious, testing the accuracy of our semantics, our experiments also showed the potential of our tool `rs2fol`. With `rs2fol`, we were able to spot cases of incompleteness in EYE's reasoning. `rs2fol` can furthermore be used to detect incorrect derivations which are far more problematic in the context of the Web where reasoning results are normally exchanged and used for applications. An entailment checker which relies on a well-established theorem prover like `rs2fol` with its underlying tool Vampire can help to prevent this problem from happening. Reasoning results can be checked independently. Developers can use `rs2fol` to evaluate their reasoning engines. `rs2fol` will furthermore help to refine the throry of RDF Surfaces. As changes to the semantics can easily be implemented in our mapping, their impact to existing use cases can be checked and the members of the community can better understand the compromises they make in terms of the semantics.

We are convinced that both of our contributions, the definition of the semantics of RDF Surfaces as well as the development of a tool for practical entailment checking will help the community bring RDF Surfaces further and to tackle the open challenges ahead of us.

References

1. Baader, F.: The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press (2007)
2. Berners-lee, T., Connolly, D., Kagal, L., Scharf, Y., Hendler, J.: N3logic: A logical framework for the world wide web. Theory Pract. Log. Program. **8**(3), 249–269 (may 2008). https://doi.org/10.1017/S1471068407003213, https://doi.org/10.1017/S1471068407003213
3. Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., Smith, M.: Owl 2 Web Ontology Language. W3C Recommendation (Dec 2012), http://www.w3.org/TR/owl2-syntax/
4. de Bruijn, J., Heymans, S.: Logical foundations of (e)rdf(s): Complexity and reasoning. In: Aberer, K., Choi, K.S., Noy, N., Allemang, D., Lee, K.I., Nixon, L., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) The Semantic Web. pp. 86–99. Springer Berlin Heidelberg, Berlin, Heidelberg (2007)
5. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1 concepts and abstract syntax, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
6. Cyganiak, R., Wood, D., Lanthaler, M.: RDF 1.1: concepts and abstract syntax (Feb 2014), http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
7. De Roo, J.: Euler Yet another proof Engine - EYE, https://josd.github.io/eye/
8. De Roo, J., Hochstenbach, P., Arndt, D.: RDF Surfaces tests (2024), https://github.com/eye reasoner/rdfsurfaces-tests

9. Ebbinghaus, H., Flum, J., Thomas, W.: Mathematical Logic. Undergraduate Texts in Mathematics, Springer New York (1996), https://books.google.de/books?id=VYLA8m7cqYcC
10. Enderton, H.B.: A mathematical introduction to logic. Elsevier (2001)
11. Harris, S., Seaborne, A.: SPARQL 1.1 query language. W3C recommendation, W3C (Mar 2013), https://www.w3.org/TR/2013/REC-sparql11-query-20130321/
12. Hayes, P.: BLOGIC (October 2009), https://de.slideshare.net/PatHayes/blogic-iswc-2009-invited-talk, ISWC 2009 invited talk
13. Hayes, P., Patel-Schneider, P.F.: RDF 1.1 Semantics. W3C Recommendation (Feb 2014), https://www.w3.org/TR/rdf11-mt/
14. Hochstenbach, P.: Latar. https://github.com/phochste/Latar, https://github.com/phochste/Latar
15. Hochstenbach, P., De Roo, J.: RDF surfaces primer (August 2023), https://w3c-cg.github.io/rdfsurfaces/
16. Hochstenbach, P., De Roo, J., Verborgh, R.: RDF surfaces: Computer says no. In: Proceedings of the 1st Workshop on Trusting Decentralised Knowledge Graphs and Web Data (May 2023), https://arxiv.org/pdf/2305.08476.pdf
17. Hochstenbach, P., van Noort, M., Arndt, D., Martens, R., De Roo, J., Bonte, P., Verborgh, R., Ongenae, F.: RDF Surfaces: Enabling Classic Negation on the Semantic Web (2024), submitted to Semantic Web Journal
18. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M., et al.: Swrl: A semantic web rule language combining owl and ruleml. W3C Member submission **21**(79), 1–31 (2004)
19. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification. pp. 1–35. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
20. Motik, B., Patel-Schneider, P.F., Cuence Grau, B.: OWL 2 Web Ontology Language Direct Semantics. W3C Recommendation (Dec 2012), https://www.w3.org/TR/owl2-direct-semantics/
21. Schneider, M.: OWL 2 Web Ontology Language RDF-Based Semantics. W3C Recommendation (Dec 2012), https://www.w3.org/TR/owl2-rdf-based-semantics/
22. Schneider, M., Sutcliffe, G.: Reasoning in the owl 2 full ontology language using first-order automated theorem proving. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction – CADE-23. pp. 461–475. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
23. Sowa, J.: Knowledge Representation: Logical, Philosophical and Computational Foundations. Brooks/Cole (2000)
24. Sowa, J.: Reasoning with diagrams and images: observation and imagination as rules of inference. Journal of Applied Logics **5**(5), 987 (2018)
25. Sutcliffe, G.: The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. Journal of Automated Reasoning **59**(4), 483–502 (2017)
26. Sutcliffe, G.: The CADE ATP System Competition. https://tptp.org/CASC/ (2024)
27. Van Herwegen, J.: Tension.js. https://github.com/joachimvh/tension.js, https://github.com/joachimvh/tension.js
28. Verborgh, R., De Roo, J.: Drawing Conclusions from Linked Data on the Web: The EYE Reasoner. IEEE Software (5), 23–27 (2015). https://doi.org/10.1109/MS.2015.63
29. W3C OWL Working Group: OWL 2 Web Ontology Language. W3C Recommendation (Dec 2012), https://www.w3.org/TR/owl2-overview/
30. Woensel, W.V., Arndt, D., Champin, P.A., Tomaszuk, D., Kellogg, G.: Notation3 language (Jul 2023), https://w3c.github.io/N3/reports/20230703/