RDF Surfaces: Enabling Classical Negation on the Semantic Web

Patrick Hochstenbach[1,2][0000−0001−8390−6171], Mathijs van Noort[2][0000−0003−2719−5402], Dörthe Arndt[3][0000−0002−7401−8487], Rebekka Martens[3], Jos De Roo[2][0000−0001−8862−0666], Ruben Verborgh[2][0000−0002−8596−222X], Pieter Bonte[4][0000−0002−8931−8343], and Femke Ongenae[2][0000−0003−2529−5477]

[1] Ghent University Library, Ghent University, Belgium
[2] IDLab, Ghent University - imec, Belgium
[3] International Center for Computational Logic, Technische Universität Dresden, Germany
[4] Department of Computer Science, KU Leuven Campus Kulak, Belgium
patrick.hochstenbach@ugent.be

Abstract. The Resource Description Framework (RDF) is a fundamental technology in the Semantic Web, enabling the representation and interchange of structured data. However, RDF lacks the capability to express negated statements in a generic way. As a result, exchanging negative information on a Web scale is thus far restricted to specific cases and predefined statements. The ability to negate (virtually) any RDF statement allows for a comprehensive way to refute, deny or otherwise invalidate claims on a Web scale. Via an intermediate step of a diagrammatic approach to logical expressions called Peirce graphs, we introduce RDF Surfaces, an extension of RDF that incorporates the concept of classic negation, known from first-order logic. Overall, RDF Surfaces provides an abstract, visual approach to negation within the Semantic Web, offering a more general and widely applicable approach than previous attempts at incorporating negation. Aside from a (traditional) programmatic syntax, RDF Surfaces can also be represented visually by means of diagrams inspired by Peirce graphs. We demonstrate negation via RDF Surfaces and how to reason upon it in illustrative use cases drawn from the domains of academic publishing and eHealth. We hope this vision paper attracts new implementers and opens the discussion to its formal specification.

Keywords: RDF Logic · Classical negation · BLOGIC · Existential graphs.

1 Introduction

Reasoning with classical negation has attracted interest in the Semantic Web since its early days. Classical negation has properties similar to processing boolean values with the `NOT` operator in computer languages. Classical negation can take "true" to "false" and vice versa. It follows the law of the excluded middle: any statement P is either "true" or "false". If some P is "true", then a double negated P is also "true" (and vice versa for "false"). This form of negation is also explicit: "false" is not a default value; it cannot be assumed; it should be stated. All other forms of negation that do not follow these principles – there are many variants – are "weaker" forms of negation. The interest in "strong" classical negation can be attributed to a desire to fulfill one of its core design principles. Just as the Web is open-ended, allowing anyone to create a webpage on a server and link to any other webpage, the Semantic Web aspires to enable anyone to express any statement about any topic [38]. The Resource Description Framework (RDF) [40] is the W3C recommendation that defines the language to express statements about anything in the universe in the form of RDF triples. Using RDF, not only Web resources can be described, but physical objects, abstract concepts, and, in general, anything that can be given an identifier. However, there are also limitations. RDF lacks the ability to express classical "strong" negation, explicitly stating negative information that also follows the law of the excluded middle and all other classical negation properties. Additionally, RDF can express existential quantification (using blank nodes): statements about *one* or more resources, but lacks the expressivity to create statements about *zero* or more resources: universal quantification. As we see below, there are reasons to desire such features, but there are also some compelling reasons why classical negation has mostly been avoided.

1.1 Why is classical negation and universal quantification desirable?

We provide three argument for classical negation and universal quantification.

First, negation and universal quantification are applicable across various use cases. Straccia and Casini [56] describe how, in medicine, it is important to distinguish between the absence of biochemical reactions between substances and not knowing about their existence, which results in a need for explicitly stating negative information. Wagner [66] makes the case for a generalized Web logic based on classic negation and quantification to drive

business processes. In an earlier paper, Wagner [65] demonstrated that two types of negation are necessary to interpret data effectively. The monotonic strong (classical) negation "Patient X did not take pill-B" expresses negative knowledge. The non-monotonic weak negation "Patient X is not registered at a hospital" expresses a negation as failure (NAF). Esteves [18] and Kebede [33] advocate for the use of negation in making policy information on permissions, prohibitions, and obligations enforceable through the Open Digital Rights Language (ODRL) [31]. They also suggest that negation can facilitate conflict resolution while creating policy documents.

Second, democratic and social reasons can be provided to desire classical negation on the Semantic Web. We live in times of massive information flows trying to influence users' social and political worldviews. Deciding what resources can be trusted, or even evaluating contrasting views, is challenging for many Web citizens. The Semantic Web is not the harbinger of absolute truth: expressing a statement as RDF does not automatically make it closer to absolute truth. Every application that processes information on the Semantic Web relies on a lesser form of truth, a relative truth: trust. By "asserting an RDF triple," a Semantic Web application assumes a triple as being "true" (whatever that means in the real world).[5] Combining multiple RDF triples forms those triples' logical conjunction (AND). Based on this relative truth of a set of triples, boolean queries can be executed using the SPARQL Query language [21], and new RDF triples can be inferred using the RDF Schema language [11]. However, without classical negation, explicitly stating what is not the case, there is no explicit notion of contradiction on the Semantic Web. This limits the applicability of formal ways to validate statements and discover contrasting views on the Semantic Web.

Third, there could be scientific reasons to add these features to the Semantic Web. Classical negation combined with existential quantification in RDF, as we will see later in the paper, provides the expressivity of first-order logic (FOL), including universal quantification. The properties of this logic have been studied over centuries. Adding classical negation to the Web, even for its own sake, would give us broad opportunities to express human reasoning and scientific inquiries. It offers the chance to present facts on the Web and the logic behind them, enhancing the scientific process's transparency.

In addition to practical reasons related to solving real-world use cases, there are also technical reasons why a classical negation and, in extension, FOL have desirable properties.

FOL retains the monotonicity property. When Web logic is monotonic, adding new asserting RDF triples to a previous set of asserted RDF triples will never invalidate previous queries or inferences. What was a valid inference using an older set of RDF triples must also be a valid inference in the updated set of RDF triples. Following Hayes (2001), Web reasoning is inherently open-ended, i.e. one can never assume all the facts about a topic are available. By consulting additional resources, new information might arise. Non-monotonic reasoning, such as NAF, assumes that information about any topic is complete. Missing information is assumed to be "false". However, this is unsafe in an open world, such as the Web, where one does not have the license to assume a statement is "false" without explicitly stating it to be "false". This is not the case for classical negation, which is explicit in the triples that should be considered "false".

There are also grounds to believe that contradictions on the Semantic Web are inherent, not detrimental as suggested by our second point, but rather benign. Hayes refers to this as the 'diamond of confusion' [24]: we may agree on a shared reality (whether it be absolute or relative truth) and a common method to create statements and express logical reasoning about this reality, yet still end up with contradictory conceptualizations of this reality. Hayes provides an example of how concepts can be conceptualized with and without a temporal dimension, which leads to contradictory results. For instance, a patient can be a fixed "thing" in one formalization with a name, address, and social security number; the same patient can be an "event in time" in another formalization where the patient with a fever is not the same patient without fever after giving a medication (a "thing" cannot both have a fever and not a fever). Understanding and identifying these contradictions among RDF resources is crucial for interpreting resources in querying and reasoning scenarios.

In some way, negation is already implicitly available in the Semantic Web when asserting triples. In logic, there are no alternative versions of "true". If, within a context, some RDF triple is regarded as being "true", this means that the negation of that triple is "false" in that context (regardless of the absolute truth or availability of other RDF triples on the open Web). Or, stated differently, within a context, any assertion negates the negated triple because $P \equiv \neg\neg P$.

1.2 Why was classical negation not added to RDF?

Considering all the reasons, why can't we incorporate this form of classic negation into RDF?

[5] In this paper we will quote "true" and "false" to remind the reader about this relative truth. However, with important logical consequences.

Monotonic logic was welcomed by many as a desirable feature on the Web, but the full expressivity of monotonic logic in the form of FOL was not. The designers of RDF deliberately chose to exclude these features, citing Lassila [39], who expressed concern that such complex features "might discourage the acceptance and adoption of RDF within the Web community."

A more compelling argument against logics with the expressivity of FOL is that they have been proven to be undecidable for finding all valid inferences from a knowledge base [58]. For machines, undecidable means that processing any arbitrary RDF with full FOL expressivity, *and* finding all correct inferences, *and* doing all of this in a finite amount of time is impossible. At most, two of these features can be achieved [47].

Finding all valid inferences in a finite time (two of the features of the previous point) is one of the core design issues in the Web Ontology Language (OWL) [9]. OWL2 DL is the flavour of OWL based on Description Logics (DL). OWL2 DL chose to use only those fragments of FOL that are decidable and (a second design choice) can be safely processed by machines in isolation. A safe execution makes computational reasoning tasks tractable (executable in polynomial time). However, two compromises need to be made to achieve this form of computability: negation and quantification need to be introduced in a weakened form, and OWL2 DL and its sublanguages need to abandon the semantics of RDF when expressing Web logic. OWL2 DL introduces "Direct Semantics" [44] as an alternative for "RDF Semantics." Only the former has the desired decidable and tractable features.[6] However, Direct Semantics disregards the RDF nature of the OWL2 DL formulas in its semantics.

1.3 Rationale for proposing the addition of classical negation and FOL expressivity to RDF

Our rationale for adding classic negation and FOL expressivity to RDF is not novel. They were articulated in Hayes' "BLOGIC" invited talk at IWSC 2009 [26]. In his argumentation, RDF is portable, i.e. any, RDF triple expresses the same data irrespective of the processing environment. However, this is not the case for current Web logics. Combinations of RDFS and OWL2 DL (and even within OWL2 DL) come with semantics that do not always agree. Web logics that use only a fraction of FOL does not commute: different fragments can disagree on what can be concluded from an RDF knowledge base. These concerns for the portability of Web logic led Hayes to propose weB LOGIC (BLOGIC) as a new approach for portable logic on the Web, rooted in the theory of *existential graphs* by Charles Sanders Peirce (1839-1914) [52]. In Hayes's vision, BLOGIC in the form of existential graphs mitigates the limited expressivity of RDF and the limitations on the portability of logic on the Web. To incorporate existential graphs, RDF needs to introduce two additional concepts: a *surface* as a boundary for negated triples and for collections of *graffiti* (blank) nodes that act as existentially quantified variables. Together with the standard assertion of triples and their conjunction, the full expressivity of FOL can be achieved without losing the structure and semantics of existing RDF resources.

Achieving BLOGIC with FOL expressivity again introduces undecidability to the Semantic Web, which is precisely what the RDF designers aimed to avoid. We argue that any *portable* Web logic in RDF will likely not be decidable. Decidability requires a delicate choice in limiting the expressive power of Web logics. While fragments of FOL may individually lead to decidability, their combination often does not, and certainly not in combination with the semantics of RDF. To communicate and transport Web logic as RDF, undecidability will be a fact we must live with and not a limiting choice.

Undecidability, in general, is not a showstopper on the Web. The satisfiability of the RDF Query language (SPARQL) is undecidable[22][69]. Billions of websites and Web applications exchange HTML, CSS, and JavaScript code in combination, fully Turing complete, thus providing a daily undecidable halting problem on the Web stack. Despite this, the Web has continued to evolve and thrive. Real-world use cases might not require solving the most extreme computability problems. In our paper, we will highlight two use cases that can be solved by implementing our translation of the BLOGIC vision in RDF, which we call *RDF Surfaces*.

We do not assume that a single machine will solve the undecidable problem and simultaneously provide a solution that can accept arbitrary knowledge input, produce all valid inferences, and always find these results in a finite amount of time. However, one of these features can be dropped to turn an undecidable problem into a decidable one where machines can assist humans in decision-making.

Our standpoint is a bit provocative. We assume that human knowledge is, by nature, undecidable and contradictory. Even in science, only romanticized views regard it as a "flawless building" of knowledge, only containing positive information and results [42]. A true Semantic Web should encompass more than just information and logic that machines can process. Human knowledge is decentralized and can be contradictory. Machine intelligence, if it wants to be in any way compared to what humans can produce, needs to be decentralized and able to be contradictory too. These contradictions should not be hidden, but expressed openly in portable syntax and logic.

[6] https://www.w3.org/TR/2012/REC-owl2-rdf-based-semantics-20121211/#Example$_o n_S emantic_D i f f erences$

In summary, a Semantic Web that is closer to enabling anyone to express any statement about any topic requires:

1. Expressing the classic negation of every possible statement.
2. Providing the full expressivity of FOL (including universal quantification).
3. Adhere to the RDF program of portability.

1.4 Contribution

As far as we know, no attempt has been made to implement classical negation, as proposed by the BLOGIC vision, in a concrete RDF syntax and implementation since Hayes' talk in 2009. Our paper will introduce RDF Surfaces as an extension of RDF's simple interpretation based on Peirce's existential graphs, with the extended semantics of classical negation and the expressivity of FOL. In this vision paper, we aim to translate Hayes' BLOGIC vision into a concrete RDF syntax, investigate the expressivity of its semantics, test the applicability in real-world use cases, demonstrate initial implementation steps, and encourage further research in formalization and potential implementations.

Our main contributions in this paper are thus as follows:

1. **RDF Surfaces Syntax:** We apply existential graphs to RDF in the form of RDF Surfaces and demonstrate how Hayes' BLOGIC vision can be made concrete with a serialization using a subset of the Notation3 syntax.
2. **Investigation of expressivity:** We demonstrate how two additions to the RDF model under simple entailment semantics provide the full expressivity of FOL with explicit quantification, with additions of (a) surfaces with negated contents, and (b) collections of graffiti (blank) nodes that act as existentially quantified variables.
3. **Showcase applicability:** We provide two use cases, one from scholarly communication and one from the healthcare domain, demonstrating the need for classic negotiation. These use cases demonstrate how positive and negative data and logic can be shared using the RDF syntax with extended semantics. Both use cases implement FOL features, such as quantification, disjunctions, and implications to share logic rules.
4. **RDF Surfaces initial reasoner implementation:** We demonstrate how RDF Surfaces and the examples of our paper can be queried using an implementation of RDF Surfaces.

1.5 Paper outline

In the remainder of this paper, Section 2 will highlight two scenarios that will be used to illustrate the application of FOL Web logic in scholarly communication and healthcare. In Section 3, we will review related work on implementing FOL and negation on the Web. In Section 4, we introduce Peirce's existential graphs. In Section 5, we introduce RDF Surfaces and its syntax, which implements Hayes' BLOGIC vision. In Section 6, an overview will be presented of four implementations of RDF Surfaces and the properties of the most mature version we used to test the use cases of this paper. In Section 7, the RDF Surfaces will be applied to the two use cases presented in Section 2. In Section 8, we will discuss our study's main points and further work. Finally, our conclusions will be presented in Section 9.

2 Running examples

This article will use two running examples to highlight the potential for negation on the Web. Our use cases help explain this paper's abstract concepts and put their application in the context of real-world use cases.

2.1 Scholarly communication

For many researchers, choosing the right place to publish is the most contentious question in their pathway to pursue an academic degree. Researchers do not need to spend much time searching online to find numerous websites from universities, libraries, publishers, and individual researchers that present the trade-offs in various publication paths. This question can be more complex than it seems. A researcher might want to give a topic more visibility by targeting a wide (nonspecialist) academic community or the general public. Alternatively, a researcher could choose to publish new results as fast as possible to claim precedence. Some scholarly communities progress in their field by sharing these fast research results as preprints in subject repositories, such as arXiv and medRxiv. However, certain journals refuse to accept research articles previously disseminated in this manner or charge high article processing charges (APC). Institutions might prefer to accept only some types of publications for satisfactory completion of a

degree, for instance, only publications from high-impact journals indexed in the Web of Science (WOS) database. The academic world could utilize library databases to share lists of journals that are explicitly stated to be part of the WOS database or explicitly state them as excluded or formally removed from coverage.

It would benefit all scholarly communication network actors to share their preferences using publicly accessible preference documents. These preferences could then provide input for smart agents to suggest new venues and provide the best advice on where to publish. An example of each actor's different types of policies is sketched below in Table 1 and Table 2.

Both the researcher and department preferences contain explicit and implicit negations. A journal that explicitly states that APC costs are not charged is a researcher's X preference. The department's Y demand for a venue in WOS excludes all venues that are explicitly not indexed in WOS. Journal facts make positive and negative facts clear. These facts can be published online by the journal or publisher's homepage or provided by library databases that track journal information. In our examples, journals ABC and DEF would be researcher and departmental preferences, but journal GHI and repository JKL would not simultaneously match the researcher and departmental preferences.

Researcher X Preferences	**Department Y Preferences**
Prefers a subject repository, a journal that does not charge APC costs or an indexed journal in WOS.	The publication venue must be indexed in WOS.

Table 1. Examples of possible publication venue preferences for a researcher and a department.

Journal ABC Facts	**Journal DEF Facts**	**Journal GHI Facts**	**Repository JKL Facts**
Indexed in WOS. Requires APC.	Indexed in WOS.	Not indexed in WOS. Does not require APC.	A subject repository.

Table 2. Facts for hypothetical journals ABC, DEF, GHI, and a repository JKL.

Medicine	**Treated affliction**	**Exclusion criteria**
High dosage of aspirin	Fever	Aspirin Allergy, Active peptic ulcer disease
Low dosage of aspirin	Acute myocardial infarction	Aspirin allergy, Active peptic ulcer disease
Beta-blockers	Acute myocardial infarction	Severe asthma, Chronic obstructive pulmonary disease

Table 3. Overview of Medicine treatment

2.2 Medicine Prescription

For the second use case, we show the applicability of FOL to the healthcare domain. Determining a suitable set of medications is a complex process, taking into account a patient's symptoms, the effectiveness of medication, the patient profile, and other factors. Specifically, the presence of allergies towards one or more types of medication, as well as possible secondary afflictions inferencing with some medication, makes for a difficult process to prescribe each patient the best suitable medication. Combining a high level of required domain knowledge and the need to perform complex reasoning upon said knowledge makes this an overall hard process to capture and automate accurately.

For a given condition, multiple possible medicines are often available to provide treatment. Here, we consider the condition of an acute myocardial infarction, with possible treatments of a low dosage of aspirin or beta-blockers. Aspirin should not be prescribed if a patient is allergic to it or suffers from active peptic ulcer disease. Likewise, beta-blockers should only be prescribed when a patient does not suffer from severe asthma or chronic obstructive pulmonary disease. Furthermore, a high aspirin dosage is known to be an effective treatment for fever, with identical exclusion criteria compared to a low aspirin dosage. The information is summarised in Table 3.

To arrive at a suitable medicine prescription, it is necessary to reason upon negative information. A patient should only be prescribed a given drug if it does hold that said drug is an effective treatment of a patient's condition and if all the known exclusion criteria of the drug are assured to *not* hold, e.g., in case of aspirin prescription, it must hold that a patient does not have an aspirin allergy nor a peptic ulcer disease.

3 Related Work

A significant body of research on defining the requirements for Web logic with negation is available with possible extensions to FOL expressivity. Table 4 presents an overview of the three main requirements we seek for a Web logic, i.e. enabling classical negation, providing the full expressivity of FOL (including universal quantification), and building upon the RDF data & syntax to attain portability, and compares them against the solutions discussed in the following paragraphs in more detail. It can be concluded from this table that none of the available solutions fit all the requirements, hence the motivation for RDF Surfaces.

Technology	Classic negation	FOL expressivity	RDF data & logic
KIF	+	+	-
Common Logic	+	+	-
N3Logic	-	-	+
FIPA	-	-	+
OWL2 DL	+ (restricted)	-	+
RIF	+	+(restricted)	-
SWRL	-	-	+
SWSL	+	+	-
De Bruijn, Tsarkov, Tammet	+	+	-
TPTP	+	+	-
Datalog	-	-	-
ASP	+	+(subset)	-
SPARQL	+(limited to filters)	+(limited to filters)	+

Table 4. Overview of technologies and their support for classic negation with explicit quantification for processing RDF data using an RDF syntax.

In the early 1990s, the Defense Advanced Research Projects Agency (DARPA) and other funding agencies started the development of the Knowledge Interchange Format (KIF) as a machine-readable interchange format of knowledge among disparate programs with an expressivity near equivalent to FOL predicate calculus including classical negation [20]. KIF's Lisp-based syntax predates the Semantic Web and was, in the 1990s, the de facto exchange format in the research community. Common Logic continued the work of KIF and has since been developed and published as the ISO standard "ISO/IEC 24707:2018" as a framework for a family of logic-based languages [32]. Common Logic can process RDF data but the syntax relies on Lisp like S-expressions. Common Logic is highly relevant for the BLOGIC vision as it includes Piercian graphical logic in the Appendix B of the ISO standard. Our paper applies this logic to the Semantic Web using an RDF syntax, ensuring compatibility with all existing RDF resources. With RDF Surfaces, we strive not to separate data and logic. No separate syntax and semantics are required to negate a set of RDF triples.

In 2000, Berners-Lee called for developing a unifying language for classical logic as an extension, or even the modification, of the RDF model. His SWeLL proposal was imaged to "allow any Web software to read and manipulate data published by any other Web software" [7]. For all logical relations to be expressed, the SWeLL project proposal advocated negation and explicit quantification as an extension to RDF. The work on SWeLL influenced the development of N3Logic [8] as an extension of RDF so that the same language can be used for transporting logic and data. N3Logic provides negation in the form of scoped negation as failure (SNAF), i.e. the monotonic version of NAF. Both NAF and SNAF are logical operations to reason about information missing from a knowledge graph, but cannot be used (or in a very limited form) to express negative information that classic negation requires explicitly. Both NAF and SNAF do not have all the desired properties of classical negation. In N3Logic, SNAF can only be used as part of an implication and not as part of the data. It is possible to create a negated statements and negated graphs by setting the consequent of an implication "false" $G \rightarrow$ `false`, but this negated graph does not have the properties of a classic negation. A double negated graph cannot be interpreted as "true".

In 2001, the FIPA RDF Content Language Specification [71] was created to specify how RDF can be used as a message content language in the communication acts of FIPA-compliant agents. It proposes a method to express negated RDF facts by adding a believe or disbelieve in the facts. The FIPA proposal adds a `fipa:Proposition` and relies on reification of RDF triples. A `fipa:believe` predicate can be added to the reified triple to express a boolean trust. Using this mechanism a single triple can be interpreted as "false", but it is not a classical negation for the same reason as N3Logic "false" is not a classical negation.

Related developments were made in 2004 with the Web Ontology Language (OWL) as an extension of RDF. The newest version, OWL 2 [9], has several profiles, of which OWL 2 DL is based on fragments of FOL. In all OWL 2 DL profiles, negation is available in the form of `owl:complementOf` and `owl:NegativePropertyAssertion`, but these negations are restricted to specific cases in the profiles to prevent the halting problem and ensure the language remains decidable. As an example, using the `owl:NegativePropertyAssertion` it is possible to state that two individual do not have the relation `:hasParent`,

```
:John a owl:NamedIndividual .
:Mary a owl:NamedIndividual .

[ a owl:NegativeObjectPropertyAssertion ;
  owl:sourceIndividual :John ;
  owl:assertionProperty :hasParent ;
  owl:targetIndividual :Mary ] .
```

From these statements it follows that `:John :hasParent :Mary` is "false", which makes it a negation of one triple. However, this negation does not have the properties of a full classical negation. The construction only works for a single triple (and not arbitrary collections of triples), and the double negation properties of classical negation are unavailable (which would lead to full FOL expressivity). As for N3Logic, a "false" is available but in a limited form. These types "false" statements can be used as constraints in an ontology, but do not have the properties of a proper classical negation. There is even a more subtle difference between this type of negation and classical negation. The John and Mary example does not state that it is impossible that John has a parent Mary, only that it is not modelled in a particular ontology.

Universal quantification is available in OWL 2 DL, but in a restricted form. The `ObjectAllValuesFrom` and `DataAllValuesFrom` predicates require a set of all individuals to choose from in an quantification. This restricted quantification differs from classical universal quantification, which permits an arbitrary (unlimited) number of individuals.

The Rule Interchange Format (RIF) [35] was an activity started in 2009 within the W3C to develop Web standards for the interchange of rules among disparate systems, especially on the Semantic Web. RIF is a collection of extensible languages and dialects serialized as XML documents. Two types of languages are available: logic-based dialects and dialects for rules with actions. The logic-based dialects include languages based on FOL and various non-FOL semantics, but do not allow negation in the rule head or body (the Horn subset). As far as we know, no RDF syntax was provided for RIF.

Semantic Web Rule Language (SWRL) [30] is a W3C membership submission that extends the OWL Web Ontology Language with the Rule Interchange Format (RIF). The proposed language extends OWL axioms to include Horn clauses for OWL descriptions and properties and a limited set of built-in functions. This form supports neither the disjunction nor the classical negation of clauses. Also, to guarantee decidability, rules are restricted to only include named individuals (and not existentially introduced individuals). The language allows for explicit quantification, but introducing variables goes beyond RDF semantics [43].

The Semantic Web Services Language (SWSL) [4] is a language for specifying the formal characterizations of Web service concepts and descriptions of individual Web services. The language consists of two sublanguages: SWSL-FOL, a full FOL language, and SWSL-Rules, a rule-based sublanguage with non-monotonic semantics. However, the authors of SWSL did not envision the need for full FOL reasoners based on SWSL-FOL as its main use case is creating Web service ontologies. The syntax of SWL is inspired by F-Logic and is not based on RDF.

Several papers by De Bruijn, Tsarkov and Tammet demonstrate the embedding of RDF in FOL using frameworks, such as F-Logic[13], Vampire[59] and JSON-LD[57]. The advantage of FOL lies in its well-established nature and the ability to define its mappings, as demonstrated by these papers. However, our project aims to achieve the expressivity of FOL within the Web language itself, specifically in the RDF model, from the ground up. The papers start from the opposite direction and try incorporating RDF semantics into a framework with FOL semantics. For similar reasons, plain FOL, such as the TPTP language[7], is excluded.

[7] https://tptp.org/Proposals/TPILanguage.html

Datalog knows some extensions which aim to incorporate negation into the rule language, most notably semi-positive Datalog and stratified Datalog [34]. Including negative atoms in rule bodies allows negation to be contained. This inclusion allows for introducing disjunctive and negative statements through a back door. Nonetheless, as is the case for SWRL considered above, Datalog's variants do not allow for existential variables, only to reason over existing ones [1]. Furthermore, many Datalog variants are restricted to *safe rules*, where all variables of a rule must occur in a positive atom of the rule body, which further restricts the free use of negative statements.

Answer Set Programming (ASP) [16] provides classical "strong" negation and NAF, and even first-order extensions can be written [41]. Eiter et al. [17] provides an extension of ASP with description logic for the Semantic Web. However, using Datalog, ASP, and even Prolog as Web logic creates a dichotomy between the world of data (RDF) and the world of logic (the computer program). In the rationale of RDF Surfaces, it should be possible to negate information in RDF and transport logic/reasoning in a portable way (using RDF). RDF Surfaces could be the language, the syntactic sugar, to transport RDF data with FOL expressivity to an ASP, Datalog, or Prolog program. We do not deny the expressivity of any of these programming languages. Our argument reverses the conventional perspective: Web logics that follow the requirements of the computing agent (decidable, tractable). RDF Surfaces advocates for Web logic for the human agent, emphasizing sharing information and logic in a portable way.

SPARQL can use classical negation and first-order expressivity in filters that can be used to query an RDF data set (that has not FOL expressivity). Three types of negation are supported: (a) the Boolean NOT operator that can be used in filters; (b) the negation as failure operators MINUS and NOT-EXISTS; and (c) a combination of OPTIONAL with the BOUND operator [2]. Additionally, SPARQL provides an RDF serialization through SPARQL-SPIN [37]. However, we regard SPARQL primarily as a query language rather than a Web logic language.

4 Existential Graphs

Existential graphs can be considered a "whiteboard" language for logical reasoning. The whiteboard surface is a logic area that contains thoughts or ideas asserted to be "true". Thoughts or ideas are written on the whiteboard in the form of symbols. These symbols can represent propositions or relations depending on the used diagram system. Groups of symbols can be encircled to create a 'nested surface' with special logical properties. Symbols and nested surfaces can be inserted and erased from the whiteboard according to a fixed set of diagram rules. These diagram rules are Peirce symbolic method of natural deduction and represent the calculus of his symbolic language. Symbol manipulation was, for Peirce, the means to make logical reasoning more natural and visual. The dynamic diagrams represent "a moving picture of the actions of the mind in thought" [53]. Peirce developed three diagrammatic systems: the Alpha system, where the symbols represent propositions and the calculus propositional reasoning; the Beta system, where the symbols represent relations and the calculus predicate logic; and the Gamma system, which explores modal and higher-order logic [54]. RDF Surfaces, the core topic of this paper, is the application of existential graphs for RDF. To guide the reader in this translation, a short introduction to the Alpha system and some highlights of the Beta system will be presented below. The application of Peirce's system to RDF will be the topic of the next section.

4.1 Default positive surface

The *default positive surface* (the whiteboard) is an area that contains zero or more symbols or deeper nested surfaces. In the Alpha system, each symbol represents a proposition. The default surface, or in Peirce's terminology the "sheet of assertion", has the property that any symbol written on it represents a formula that is considered logically "true" (that is, "true" in the relative sense, not an absolute truth). The order in which symbols are written or their position on the surface has no special meaning. The positive surface is interpreted as the logical conjunction ($\wedge$) of all symbols and surfaces written on it. If the symbols A and B are written on the positive surface, then the conjunction $A \wedge B$ is interpreted as true by that surface. The empty positive surface is interpreted as a tautology ("true" in every possible interpretation)

4.2 Negative surface

A *negative surface* (in Peirce's terminology, the "cut") has the property that any symbol that is written on it represents a formula that is considered logically "false". If the symbols A and B are written on a negative surface, then the conjunction $A \wedge B$ is interpreted as being "false" by that surface. Or stated differently, the negation $\neg(A \wedge B)$ is "true" on the default positive surface on which the negative surface is written. A negative surface is the classical negation ($\neg$) of the symbols written on it. An empty negative surface is interpreted as a contradiction ("false" in every possible interpretation).

4.3 Nested surfaces

Negative surfaces can be nested inside other negative surfaces, but no nested surfaces are allowed to overlap another. We will define the *surface nesting level* as the number of "negative borders" one must cross to reach a surface's interior. The *parity* of the surface nesting level is the number of crossings modulo 2. We will see that the surface nesting level and its parity will be vital for the diagram manipulation rules, which we will explain in the next subsection. For any surface S, the *containment* is defined as the set of all symbols/nested surfaces enclosed by S, including the surface S itself.

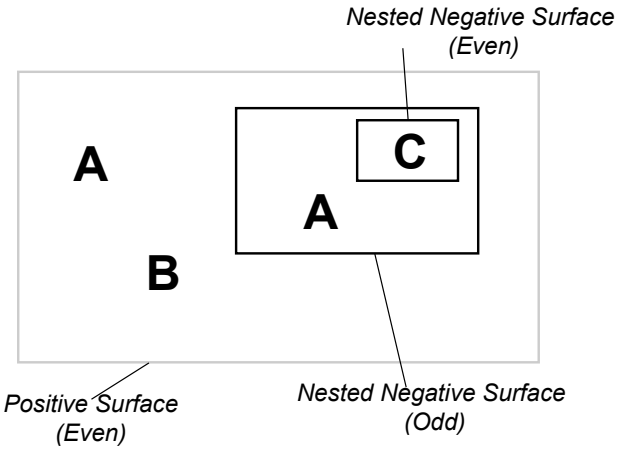


Fig. 1. A positive surface with the symbols A and B, a nested negative surface with the symbol A, and a deeper nested negative surface with the symbol C. The logical interpretation of this diagram is $A \wedge B \wedge \neg(A \wedge \neg C)$. The parity of the surface is the number of "negative borders" one needs to cross to reach the symbols modulo 2. The positive surfaces has parity 0, the negative surfaces with A parity 1, and the negative surface with C parity 0.

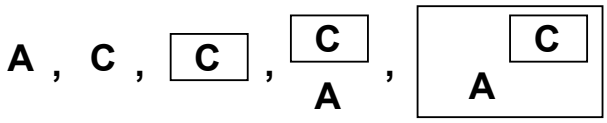


Fig. 2. The first inner nested negative surface (the one with odd parity) of Figure 1 has a containment of 5 symbols/nested surfaces.

Positive surfaces, negative surfaces, and nesting are all required to interpret any composite diagram. As an example, in Figure 1, we see a default positive surface with propositions A and B, a nested negative surface with proposition A, and inside the nested surface another nested surface with proposition C. On both the default positive surface and the nested negative surface, A represents the same proposition. The positive surface has parity 0, the first nested negative surface has parity 1, and the innermost negative surface has parity 0. The full interpretation of this diagram is $A \wedge B \wedge \neg(A \wedge \neg C)$. The containment of the first (outer) negative surface in Figure 1 includes 5 symbols/nested surfaces as is shown in Figure 2.

4.4 Diagram rules

Peirce provided for his Alpha system four diagram manipulation rules to insert and erase symbols and nested surfaces:

- **R1 Insertion**: Any symbol/nested surface can be introduced on any surface with parity 1.
- **R2 Erasure**: Any symbol/nested surface can be erased on any surface with parity 0.
- **R3 Double Cut**: A double nested surface can be replaced by its interior when the outer region is empty.

– **R4 (De)iterate**: Any symbol/nested surface S' on a surface S can be placed or erased from any surface that is not part of S', but contained by S.

The last rule **R4** requires some explanation. A copy of any symbol/nested graph can be added or erased from any nesting level. Starting from Figure 1 we can create Figure 3 by adding the B symbol in the nested surface with party 1, and the inner nested surface with party 0. Likewise, the A symbols can be added at any nesting level. Rule **R4** also erases these A and B copies. Rule **R4** does not allow to place a copy of a nested surface within itself.

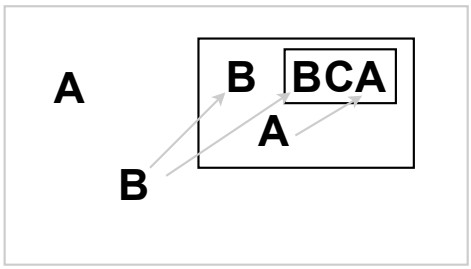


Fig. 3. Using rule **R4** a copy of a symbol or nested surface can be placed in at any surface level which is contained by the origin surface. But, it is not possible to place a copy of a nested surface within itself.

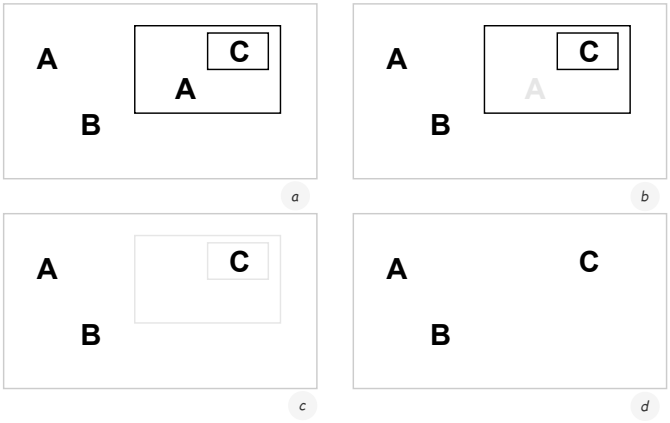


Fig. 4. (a) A representation of $A \wedge B \wedge \neg(A \wedge \neg C)$; (b) the deiterate rule **R4** is applied to erase a copy of A from the nested negative surface; (c) the double cut rule **R3** is applied to erase the double nested surface; (d) the result of the deduction $A \wedge B \wedge C$.

Diagram manipulation rules can be applied to create new diagrams that maintain the same logical truth value as the original diagram. For instance, Figure 4a is a copy of Figure 1. By applying the deiterate rule **R4** to this diagram, we can remove the A copy from the nested negative surface, which results in Figure 4b. As a next step, we see that Figure 4b contains a double nested negative surface, which can be erased using the double cut rule **R3**. This results in Figure 4c and eventually Figure 4d. This symbolic calculus provides a deduction for Equation (1).

$$A \wedge B \wedge \neg(A \wedge \neg C) \overset{R4}{\vdash} A \wedge B \wedge \neg(\neg C) \overset{R3}{\vdash} A \wedge B \wedge C \tag{1}$$

An implication can be recognized in Equation (1). A negative surface containing A plus a deeper nested negative surface containing C is the diagrammatic equivalent of the logical formula $\neg(A \wedge \neg C)$, which is, by definition, the implication: $\neg(A \wedge \neg C) \equiv A \rightarrow C$. That is, it cannot be the case that A is "true" and C is not "true". Stated differently, from A follows C. This can be written in symbolic form as:

$$A \wedge B \wedge \neg(A \wedge \neg C) \vdash A \wedge B \wedge (A \rightarrow C) \vdash A \wedge B \wedge C \tag{2}$$

The correspondence between Alpha calculus and propositional logic (Beta calculus with predicate logic) is not coincidental. Zeman formally established it in the early 1960s [68]. Important for our discussion is that any compound truth-functional statement can be written diagrammatically using symbols and (nested) negative surfaces:

- The logical conjunction $\wedge$ is given by writing symbols (or nested negative surfaces) on a positive surface (Figure 5a).
- The logical negation $\neg$ is given by writing symbols (or nested negative surfaces) on a negative surface (Figure 5b).
- The logical disjunction $\vee$ is given by noting that $A \vee B \equiv \neg(\neg A \wedge \neg B)$ (Figure 5c).
- The logical implication $\rightarrow$ is given by noting that $A \rightarrow B \equiv \neg(A \wedge \neg B)$ (Figure 5d).

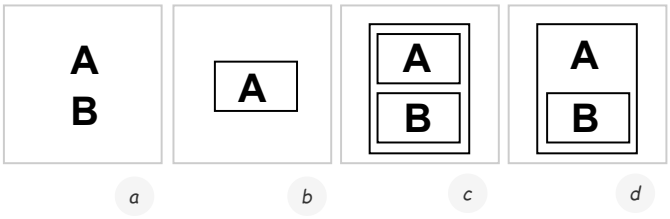


Fig. 5. Diagram representations of (a) $A \wedge B$, (b) $\neg A$, (c) $A \vee B$, and (d) $A \rightarrow B$

4.5 Quantification

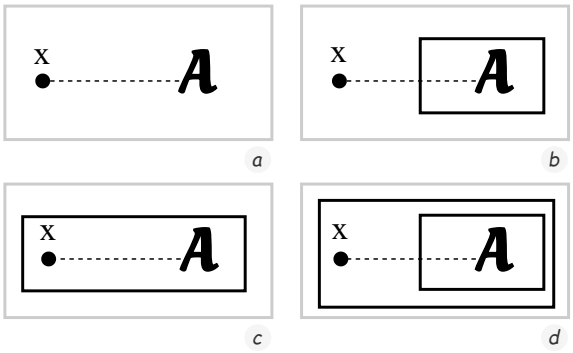


Fig. 6. Quantification using the Beta system. The interpretations of these diagrams are: (a) $\exists x : \mathcal{A}(x)$, (b) $\exists x : \neg\mathcal{A}(x)$, (c) $\neg(\exists x : \mathcal{A}) \equiv \forall x : \neg\mathcal{A}(x)$, and (d) $\neg(\exists x : \neg\mathcal{A}) \equiv \forall x : \mathcal{A}(x)$.

The diagrams thus far addressed reasoning with propositions using the Alpha system. The Beta system expands the Alpha system by adding quantification and the logic of relations. In this article, we will only provide a concise example of the Beta system, providing enough detail to understand the quantification techniques of the next section.

A (possibly indexed) heavy dot $\bullet$ denotes that some identity exists in the Beta system. These heavy dots can be extended into a "line of identity", which has the same interpretation as the heavy dot. When two or more heavy dots appear in a diagram with indexes with equal labels, they act as coreferences and denote the same identity. In the Beta system, symbols are interpreted as relations. A relation's *arity* is the number of "lines of identity" attached to the symbol. In our examples, we will use a script-like font to differentiate relation symbols from proposition symbols.

In Figure 6, we see a Beta diagram that means "There exists some entity x which is an $\mathcal{A}$" or as a logical formula $\exists x : \mathcal{A}(x)$. If the relation symbol is written on a negative surface, as in Figure 6b, the interpretation becomes "There exists some entity x which is not an $\mathcal{A}$" or as a logical formula $\exists x : \neg\mathcal{A}(x)$. When the heavy dot is placed inside the nested surface, as in Figure 6c, the contents of the surface and the heavy dot will be denied. This diagram means

"It is false that some entity x is an $\mathcal{A}$," which is equivalent to "Nothing is an $\mathcal{A}$," or as a logical formula $\forall x : \neg\mathcal{A}(x)$. When we move the relation to a deeper nested negative surface, as in Figure 6d, we get the interpretation "It is false that some identity x is not an $\mathcal{A}$," or stated differently "Every entity x is an $\mathcal{A}$", which can be written as the logical formula $\forall x : \mathcal{A}(x)$.

De Morgan's identity, Equation (3), can be recognized in these four examples. A heavy dot on a surface with parity 0 (e.g., the default positive surface) should be interpreted as an existentially quantified variable. A heavy dot on a surface with parity 1 should be interpreted as a negated existentially quantified variable equivalent to a universally quantified variable.

$$\neg(\exists x : \mathcal{P}(x)) \equiv \forall x : \neg\mathcal{P}(x) \tag{3}$$

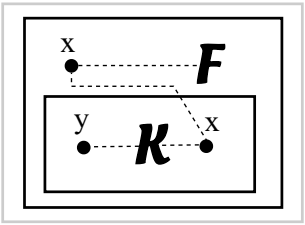


Fig. 7. The interpretation of the quantified variables x and y depend on the surface nesting level. The outermost x is on the negative surface with nesting level 1, thus, its scope is the outer surface. The y is on the negative surface with nesting level 2, and its scope is the inner surface.

Using the Beta system, arbitrary complex statements with predicates can be composed of (indexed) heavy dots (extended into "lines of identity") and relation symbols. Figure 7 expresses the implication "It is false that some entity x is famous ($\mathcal{F}$) and not some entity y knows ($\mathcal{K}$) x". That is, "For any entity x that is famous, there is some entity y that knows x." Or, stated as a logical formula:

$$\neg(\exists x : \mathcal{F}(x) \wedge \neg(\exists y : \mathcal{K}(y, x))) \equiv \forall x \exists y : \mathcal{F}(x) \rightarrow \mathcal{K}(y, x) \tag{4}$$

The scope of the heavy dot depends on the surface with the lowest surface nesting level on which the (possibly indexed) heavy dot is written. In Figure 7, the heavy dot with index x is placed on the negative surface with level 1 and extended to the negative surface with level 2.

For the Beta system diagram, rules can be provided that are an extension of the four Alpha system diagram rules that were described in Section 4.4, by including rules for heavy dots. For the remainder of this paper, we do not delve into the specifics of this extension. Instead, we refer the reader to Roberts [50] for a thorough introduction. It is sufficient to state that a heavy dot is treated as a wildcard.

In Figure 8, we see an example of a Beta system derivation using heavy dots. Figure 8a is a copy of Figure 7, which includes on the default positive surface the statement $\mathcal{A}...\mathcal{F}$ that symbolizes: "There is some A which is famous". In Figure 8b, we see that a heavy dot was applied as a wildcard: $x...\mathcal{F} = \mathcal{A}...\mathcal{F}$ if $x = \mathcal{A}$. This replacement $x = \mathcal{A}$ must also be done for the x in the deeper nested negative surface. In Figure 8c, we apply the deiteration rule **R4** to remove the copy $\mathcal{A}...\mathcal{F}$ from the nested negative surface. From Figure 8c, we get to Figure 8d by applying the double cut rule **R3**.

It should be noted that the Beta system does not have a notion of logical constants. The diagram notation $\mathcal{A}...\mathcal{F}$ should be interpreted as:

$$\exists x : \mathcal{A}(x) \wedge \mathcal{F}(x) \tag{5}$$

In the Beta system, relations with any arity can be introduced on the whiteboard (the $\mathcal{A}$ and $\mathcal{F}$ in our examples were of arity 1 and $\mathcal{K}$ of arity 2). We will see in the next section on RDF Surfaces we use Peirce's Existential graphs as an inspiration for applying surface language in RDF context, but we will deviate from the Beta system and allow constants, use only relations of arity 2, and reinterpret the heavy dot as the blank nodes of RDF.

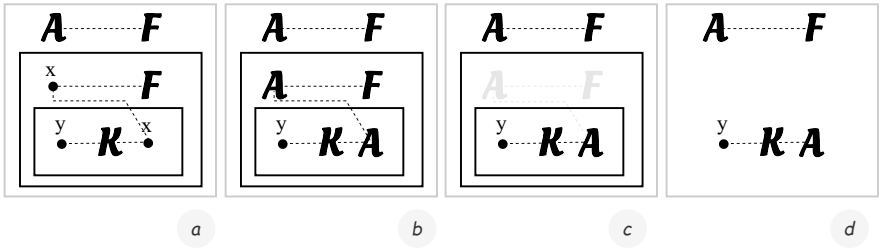


Fig. 8. Concise example of a derivation using the Beta system: (a) the original diagram, (b) x is a wildcard which matches A on the positive surface, (c) using the deiterate rule **R4** the copy $\mathcal{A}...\mathcal{F}$ can be removed from the nested negative surface, (d) using the double cut rule **R3** the double nested surface can be removed.

5 RDF Surfaces

In the previous section provided the foundation for expressing FOL using Peircian diagram logic. This section applies Peirce diagrams to the Semantic Web. As an introductory step, we add RDF triples in the form of an implication in a Peirce diagram and demonstrate that the same diagram rules of Section 4 can be applied to derive new RDF triples. In a second step, we translate the diagrams into an RDF syntax and semantics we named *RDF Surfaces*.

With existential graphs, we are not limited to including only abstract symbols in the diagrams. In Figure 1, the A, B, and C can be replaced by any concrete object, such as ideas, drawings, and even RDF statements and still retain the same expressive power of visualized reasoning. Using these insights, we use the results of our previous section to apply existential graphs to RDF following the BLOGIC vision of Hayes. This new form, called RDF Surfaces, will demonstrate how the expressivity of FOL can be implemented with only two additions to the RDF Model: *surfaces* on which to draw RDF Graphs and the explicit scoping of quantified variables using collections of blank nodes, called the *graffiti*, which are the "heavy dots" of the previous section.

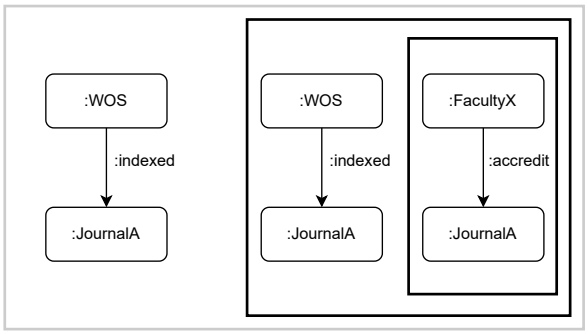


Fig. 9. A diagram representations of the RDF fact ⟨`:WOS :indexed :JournalA`⟩ and the implication ⟨`:WOS :indexed :JournalA`⟩ → ⟨`:FacultyX :accredit :JournalA`⟩.

To illustrate our approach, Figure 9 presents an example where the previous section's abstract symbols A, B, and C have been replaced with RDF statements. The default surface in our example contains the diagram:

$$\boxed{\texttt{:WOS}} \overset{\texttt{:indexed}}{\rightarrow} \boxed{\texttt{:JournalA}}$$

This diagram stands for the RDF triple ⟨`:WOS :indexed :JournalA`⟩, i.e. "WOS added JournalA to its index". It also contains an implication with the meaning "If WOS indexed JournalA, then FacultyX accredited JournalA". As such, Figure 9 expressed in symbolic form represents:

$$\begin{aligned} &\langle \texttt{:WOS :indexed :JournalA} \rangle \wedge \\ &\big(\langle \texttt{:WOS :indexed :JournalA} \rangle \rightarrow \langle \texttt{:FacultyX :accredit :JournalA} \rangle\big) \end{aligned} \tag{6}$$

Using the diagram rules from Section 4.4 we can deduce ⟨`:FacultyX :accredit :JournalA`⟩ in two steps:

- **R4** : deiterate `:WOS` $\overset{\texttt{:indexed}}{\rightarrow}$ `:JournalA` from the nested negative surface with nesting level 1.
- **R3**: remove the double cut that was created by the first step.

To make RDF Surfaces more concrete, we define the concept of a *Hayes triple* and a *Hayes graph.*

Definition 1. *A Hayes triple consists of three components* ⟨***Gr S H***⟩ *where:*

- ***Gr*** *is a (possibly empty) set of blank nodes, called graffiti.*
- ***S*** *is a surface type.*
- ***H*** *is a (possibly empty) set of RDF triples and Hayes triples, called the Hayes graph.*

Following Viswanathan [64], we can define a scope, plus bound and free occurrences of graffiti nodes, as follows:

Definition 2. *For the Hayes triple* ⟨***Gr S H***⟩, ***H*** *is said to be the scope of the graffiti nodes in* ***Gr***.

Definition 3. *Every occurrence of a graffiti node g in* ***Gr*** *as a blank node in any (nested) RDF triple of* ***H*** *is called a bound occurrence of g in* ***H***. *The blank node in that RDF triple is a coreference to g. Any occurrence of a graffiti g that is not bound is called a free occurrence of g in* ***H***.

We say that the graffiti nodes of ***Gr*** are *on* the surface defined by the Hayes graph ***H***. The recursive definition of a Hayes graph reflects the fact that, following Section 4, we deal with nested surfaces. The surface type translates to different surface interpretations we saw in Section 4, i.e. positive and negative. In general, more surface types can be imagined. Hayes BLOGIC presentation, for example, mentioned neutral and deprecated surface types as possible extensions. Neutral surfaces would not be asserted or negated and could be used for packaging RDF triples without giving them a truth value. Deprecated surfaces could be used to provide a time constraint on the truth value of a surface.

Graffiti, in the form of blank nodes, are a direct translation of the heavy dots of the previous section. Blank nodes that appear in the RDF triples of ***H*** act as coreferences to the graffiti nodes ***Gr*** with the same label on an ancestor surface. These graffiti nodes are scoped in ***H*** in such a way that if a deeper nested Hayes graph exists that contains graffiti nodes using the same label as a parent Hayes graph, this deeper nested Hayes graph creates a new logical scope for the graffiti node with that label. If a blank node in a RDF triple does not have such a coreference to a graffiti node, it is said to be free.

Figure 10 provides an example. A graffiti node with label `_:B` is written on the negative surface with parity 1. There are also blank nodes written on the same surface, and on a deeper nested negative surface with parity 0 with the same label `_:B`. All these blank nodes, labeled `_:B`, are coreferences to the graffiti node with label `_:B`.

Given the concept of a Hayes graph we can define an RDF Surface.

Definition 4. *An RDF Surface is a Hayes triple with a positive surface type, and every Hayes triple nested within the RDF Surface has a negative surface type (regardless of the depth of their nesting).*

This definition reflects the role of the default surface as a positive surface, i.e. its content has the value "true". All the nested surfaces are negative, i.e. its content has the value "false".

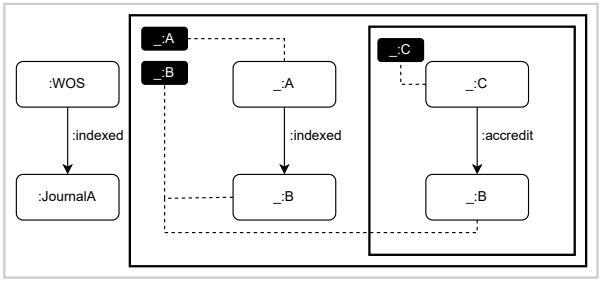


Fig. 10. A diagram representation of the fact `:WOS :indexed :JournalA` and an implication which means "Every `A` that indexed `B`, has some `C` that accredits `B`".

Each blank node that occurs somewhere in an RDF Surfaces graph, i.e. either in the Hayes graph of the positive surface or in that of a nested Hayes graph, but not in any of the sets of graffiti of its parent Hayes graphs, is considered *free*. We subsequently 'existentially close' the RDF Surfaces graph by adding graffiti nodes on the top level, the default positive surface, for every freely occurring blank node. In the case of identically named blank nodes, there is only one single graffiti to which all occurrences are referenced. This coincides with the approach taken in plain RDF, i.e. any blank node in RDF is implicitly assumed to be existentially quantified. Within RDF Surfaces, we aim to support any existing RDF document. To achieve this, we create an implicit existential closure by capturing these seemingly 'free' blank nodes within RDF in the set of graffiti of the top-level, positive Hayes graph, 'pinning down' their existential nature.

An example of a full RDF Surfaces graph using explicit quantification can be seen in Figure 10. The default surface contains the assertion: ⟨`:WOS :indexed :JournalA`⟩. The right part of the figure contains an implication similar to the one in Figure 9, but now using the graffiti blank nodes (in black) to denote quantifications that are used in RDF Surfaces. The graffiti nodes `_:A` and `_:B` are attached to the negative surface with nesting level 1, parity 1. This position of the graffiti nodes `_:A` and `_:B` on a negative surface with parity 1 provides an interpretation of these nodes as *existential* quantified variables that can be transformed into *universal* quantified variables. However, the graffiti node `_:C` is attached to a negative surface with nesting level 2, parity 0. The position of the graffiti node `_:C` on this negative surface with parity 0 interprets the node as an *existential* quantified variable. All other blank nodes (in white) on the negative surface and its deeper levels are coreferences to the graffiti nodes with the same label. The meaning of implication in Figure 10 is "Every `A` that indexed `B`, has some `C` that accredits `B`". The full diagram can be symbolized using Equation (7), as follows:

$$\begin{aligned} &\langle \texttt{:WOS :indexed :JournalA} \rangle \wedge \\ &(\forall a, b: \ \exists c : \langle a \ \texttt{:indexed} \ b \rangle \rightarrow \langle c \ \texttt{:accredit} \ b \rangle) \end{aligned} \tag{7}$$

Applying the diagram rules of Section 4.4 in combination with the extension for quantification in Section 4.5, the following deduction is possible:

- `_:A` —:indexed→ `_:B` can be deiterated using rule **R4** when `_:A` is equal to `:WOS`, and `_:B` is equal to `:JournalA`.
- This leaves us with a double cut that can be removed using rule **R3**.
- The result is a graffiti node labeled `_:C` on the default positive surface and the new fact: `_:C` —:accredit→ `:JournalA` .

When multiple facts are available, the diagram rules from Section 4.4 can be repeatedly applied to make derivations, which always result in the same graffiti nodes labeled `_:C`. Using the iteration rule **R4** we can add many copies of the implication of Figure 10.

5.1 Notation3 serialization

Hayes introduced an annotated Turtle syntax, using comments, in his BLOGIC presentation to express Peirce's diagrams. With RDF Surfaces we opted to use a subset of the Notation3 (N3) [67] syntax to provide a more structured serialization format in RDF itself. N3 already has parsers, such as EYE [14] and n3 [62], that provide syntactical support for graph terms and RDF lists as first-class citizens of the language, which provide a possibility for a direct translation of RDF Surfaces diagrams into N3. Graph terms can be used to describe a RDF/Hayes graph $\boldsymbol{H}$ on a surface and lists for the set $\boldsymbol{Gr}$ of graffiti on a surface.

It is important to differentiate between the requirements for the RDF model (and its semantics) to describe RDF Surfaces and the serialization format to transport that model.

Our first choice for using N3 as a serialization format is pragmatic, based on de facto approaches to make statements about sets of RDF triples. Alternatively, we could have considered named graphs. However, if the surfaces have IRIs, this approach risks self-referential surfaces and paradoxes. If the surfaces use blank node identifiers, it also leads to issues with the quantification of surfaces. We aimed to stay close to the Semantic Web ideal that everything can be expressed as triples.

Our requirements for the RDF model focus on identifying the minimal additions needed to achieve the expressivity of FOL with explicit quantification. RDF Surfaces requires two additions: a concept of a surface containing a (possible empty) set of triples, and a concept of a collection of graffiti blank nodes that act as existentially quantified

variables. RDF Surfaces relies on the simplest version of entailment: only Simple Entailment is required[8] and other entailments such as RDFS follow by applying RDF Surfaces formulas. That is, from:

$$\langle \texttt{:CelestialObject rdfs:subClassOf :Cheese} \rangle \wedge \langle \texttt{:Moon a :CelestialObject} \rangle$$

one may conclude

$$\langle \texttt{:Moon a :CelestialObject} \rangle$$

but not

$$\langle \texttt{:Moon a :Cheese} \rangle,$$

as this relies on RDFS entailment. In RDF Surfaces, we build RDF entailment from the ground up, creating a foundational layer to make entailments explicit, based on FOL. To entail the RDFS version, an extra formula can be introduced that explains what `rdfs:subClassOf` means, such as the RDF Surfaces version of the symbolic form:

$$\forall x, y, z : (\langle \texttt{x rdfs:subClassOf y} \rangle \wedge \langle \texttt{z a x} \rangle) \rightarrow \langle \texttt{z a y} \rangle.$$

To allow for a serialization of surfaces and graffiti we use only a subset of the N3 *syntax* (not its semantics):

- The Turtle textual syntax for RDF: meaning that any valid RDF 1.1 Turtle [6] graph is a valid RDF Surfaces graph.
- N3 list terms as a set of graffiti (blank) nodes.
- N3 graph terms as a conjunction of quoted statements.
- A new IRI `log:onNegativeSurface` which indicates the type of surface (currenly only the negative surface type is used).

A (default) positive surface is implicit assumed on the boundary of every RDF document. That is, every RDF triple in existing RDF documents is assumed to be "on" a (default) positive surface. Every 'free' blank node in an RDF graph is assumed to be implicit existential closed in the set of graffiti nodes of the (default) positive surface. These graffiti nodes cannot be shared between RDF documents. When combining two or more RDF documents, a new set of graffiti nodes must be created ("engraved" in Hayes terminology) in the resulting RDF document.

A Hayes triple is expressed in "RDF Surfaces in N3" (N3S) as a Turtle/RDF triple with on the subject position an N3 collection to represent the graffiti nodes that are "on" a surface. In the predicate position, we write the surface type: in our case `log:onNegativeSurface`. In the object position, an N3 graph term represents the Hayes graph. The usage of N3 graph terms is restricted so that they only occur in the object position of a triple having a surface type as a predicate. N3 collections (lists terms) are first-class citizens in N3 (and N3S). We require this feature in the N3 serialization of RDF Surfaces because "classical" RDF lists introduced blank nodes. This is problematic, as we need to exactly know where blank nodes are quantified.

Listing 1.1 provides an example of such a N3S serialization of the RDF Surfaces diagram of Figure 10. The triple on line 4 is "on" the (default) positive surface. Line 6 defines a negative surface with the graffiti nodes `_:A` and `_:B`, interpreted to be *on* the negative surface. The blank nodes `_:A` and `_:B` on lines 8 and 11 act as coreferences to the graffiti nodes declared by the outer negative surface. Line 10 defines a nested negative surface with one graffiti node, `_:C`. The blank node `_:C` on line 11 is a coreference to the graffiti node declared on line 10.

```
1  @prefix log: <http://www.w3.org/2000/10/swap/log#> .
2  @prefix : <https://example.org/ns#> .
3
4  :WOS :indexed :JournalA .
5
6  ( _:A _:B ) log:onNegativeSurface {
7
8      _:A :indexed _:B .
9
10     ( _:C ) log:onNegativeSurface {
11         _:C :accredit _:B .
12     } .
13 } .
```

Listing 1.1. The N3S serialization of the RDF Surfaces diagram depicted in Figure 10.

[8] https://www.w3.org/TR/rdf11-mt/#simpleentailment

5.2 Blank nodes and explicit scoping of logical variables

Using the concept of scope, as defined by Definition 2, a careful relabeling of graffiti nodes in Listing 1.1 without changing the meaning of the RDF Surfaces graph is possible.

```
1  @prefix log: <http://www.w3.org/2000/10/swap/log#> .
2  @prefix : <https://example.org/ns#> .
3
4  :WOS :indexed :JournalA .
5
6  ( _:A _:B ) log:onNegativeSurface {
7
8      _:A :indexed _:B .
9
10     ( _:A ) log:onNegativeSurface {
11         _:A :accredit _:B .
12     } .
13 } .
```

Listing 1.2. A relabeled version of Listing 1.1 with the same meaning, using the scoping of graffiti nodes.

In Listing 1.2, a graffiti node with label *_:A* appears on lines 6 and 10, with corresponding blank nodes on lines 8 and 11. The question remains: which graffiti nodes on lines 6 and 10 should bind? RDF Surfaces uses the following *binding convention*:

Convention: Blank nodes bind to graffiti nodes with the same label on the closest parent surface.

In our example, this would imply that the blank node with label `_:A` on line 8 is bound to the graffiti node with the same label declared on line 6, and the blank node with label `_:A` on line 11 is bound to the graffiti node with the same label declared on line 10. This convention makes the meaning of Listing 1.2 equal to Listing 1.1.

The binding of blank nodes that are *not* mentioned in any graffiti is implicit and becomes 'existentially closed' on the top-level (default) positive surface. In this sense, RDF Surfaces mimic the behavior of plain RDF. Well-known ambiguities for quantification can be avoided [29]. For example, the design documents for N3Logic added syntactical features, such as `@forAll` and `@forSome` for quantification, but because graphs have no order in RDF, ambiguity exists in which N3 formulas are quantified by these quantifiers [3]. In RDF Surfaces, quantification is unambiguous using mandatory graffiti nodes with scope.

5.3 Explicit negation

Within the boundaries of RDF Surfaces, positive or negative facts can be stated without requiring access to all possible RDF triples in a much bigger world. For instance, to express that an RDF triple in an RDF Surfaces graph does not have a particular property, we can state (Listing 1.3):

```
1  @prefix log: <http://www.w3.org/2000/10/swap/log#> .
2  @prefix : <https://example.org/ns#> .
3
4  () log:onNegativeSurface {
5      :WOS :indexed :JournalABC .
6  } .
```

Listing 1.3. An RDF Surfaces graph with an explicit negation with meaning "WOS did not index JournalABC."

The meaning of Listing 1.3 is: "It is not the case that `:WOS :indexed :JournalABC`". If this RDF Surfaces graph would contain any triples stating that `:WOS :indexed :JournalABC`, or if this could be inferred by deduction, it would be in conflict with the stated negation and thus create a contradiction. These contradictions are explicitly available by the semantics of RDF Surfaces and are not hidden as in the current RDF model.

Adding explicit negation as data or as a consequence of an implication makes it possible to state which statements are "false". This explicit negation is not equivalent to NAF. One could be tempted to create a negation to simulate NAF as in Listing 1.4. However, Listing 1.4 does not entail `:Surface :is :JournalLess`. The RDF Surfaces graph, as shown in Figure 11, does not contain the *explicit (negative) fact* that some entity is *not* a journal. The deiteration rule **R4** cannot be applied.

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix : <https://example.org/ns#> .

:BookABC a :Book .
:BookDEF a :Book .

(_:A) log:onNegativeSurface {

    () log:onNegativeSurface {
        _:A a :Journal .
    } .

    ( ) log:onNegativeSurface {
        :Surface :is :JournalLess .
    } .
} .
```

Listing 1.4. A negative surface **cannot** be used as a negation as failure. The triple ⟨`:Surface :is :JournalLess`⟩ is **not** a logical consequence of this RDF Surfaces graph.

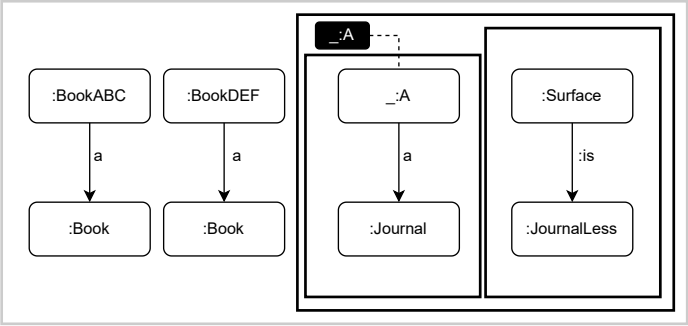


Fig. 11. A diagram representations of Listing 1.4. There is no negative fact on the default positive surface to deiterate the negation of ⟨ `_:A a :Journal` ⟩. It is **not** possible to conclude ⟨ `:Surface :is :JournalLess` ⟩.

Explicit negation is also distinct from SNAF, which is available in Notation3 by the `log:notIncludes` builtin predicate[9]. Both NAF and SNAF styles of negation make statements about information that cannot be derived from a knowledge base. NAF requires assumptions about the rest of the world, thus implying complete knowledge about missing information. SNAF requires assumptions about missing triples within the boundaries of an RDF document. In contrast, explicit negation does not require any assumptions about missing information. Instead, RDF Surfaces can explicitly state negative information or derive explicit negative information.

5.4 Disjunctions in the data, antecedent and consequent of an implication

Following the hints in Section 4.4, a disjunction can be constructed using a combination of negations with conjunctions given that $A \vee B \equiv \neg(\neg A \wedge \neg B)$. Listing 1.5 provides a fictional example of such disjunction by adding nested negative surfaces in a negative surface. Expressed in a symbolic form this becomes:

$$\forall x \exists y : \langle \texttt{x a :JournalArticle} \rangle \vee (\langle \texttt{x a :Preprint} \rangle \wedge \langle \texttt{y :reviewed x} \rangle)$$

which is equivalent to:

$$\neg\left(\exists x : \neg\langle \texttt{x a :JournalArticle} \rangle \wedge \neg\left(\exists y : \langle \texttt{x a :Preprint} \rangle \wedge \langle \texttt{y :reviewed x} \rangle\right)\right).$$

[9] https://w3c.github.io/N3/reports/20230703/builtins.html#log:notIncludes

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix : <https://example.org/ns#> .

( _:X ) log:onNegativeSurface {

    ( ) log:onNegativeSurface {
        _:X a :JournalArticle .
    } .

    ( _:Y ) log:onNegativeSurface {
        _:X a :Preprint .
        _:Y :reviewed _:X.
    } .
} .
```

Listing 1.5. Adding nested negative surfaces in a negative surface creates a disjunction. The RDF Surfaces graph means, "Any article is a journal article or a preprint, and some S reviewed it, or both."

Listing 1.5 can be read in many ways, all equally valid, and all share the same meaning:

- Everything is a journal article or a preprint that was also reviewed by someone or both.
- If there is something that is not a journal article, it is a preprint, and someone reviewed it.
- If there is something that is not a preprint that someone reviewed, then it is a journal article.

This is because $\neg X \rightarrow Y \equiv \neg Y \rightarrow X \equiv X \vee Y$. RDF Surfaces makes the equivalence of all these readings explicit.

To illustrate the effect of disjunction in reasoning, a negation can be added on the default positive surface, which, for instance, expresses that $\langle$`:MyArticle a :JournalArticle`$\rangle$ is "false":

```
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
@prefix : <https://example.org/ns#> .

( ) log:onNegativeSurface {
    :MyArticle a :JournalArticle .
} .
```

Applying the diagram rules of Section 4.4 the following deduction is possible:

- `() log:onNegativeSurface { _:X a :JournalArticle }` can be deiterated from Listing 1.5 using rule **R4** when `_:X` is equal to `:MyArticle`.
- This leaves us with a double cut that can be removed using rule **R3**.

As result, the deductive closure of Listing 1.5 contains the RDF triples:

```
@prefix : <https://example.org/ns#> .

:MyArticle a :Preprint .
_:Y :reviewed :MyArticle .
```

The diagram form of Listing 1.5 and the diagrammatic derivation is available in Figure 12.

Disjunctions are not only available in the data but can also be added to the antecedent and consequent of an implication. That is, both forms are possible:

$$\forall x : (\langle \texttt{x a :Journal} \rangle \vee \langle \texttt{x a :Book} \rangle) \rightarrow \langle \texttt{x a :Publication} \rangle$$

and

$$\forall x : \langle \texttt{x a :Publication} \rangle \rightarrow (\langle \texttt{x a :Journal} \rangle \vee \langle \texttt{x a :Book} \rangle) .$$

Disjunctions in the consequent is a feature in RDF Surfaces unavailable in N3Logic. The latter implication could be written in RDF Surfaces using Listing 1.6.

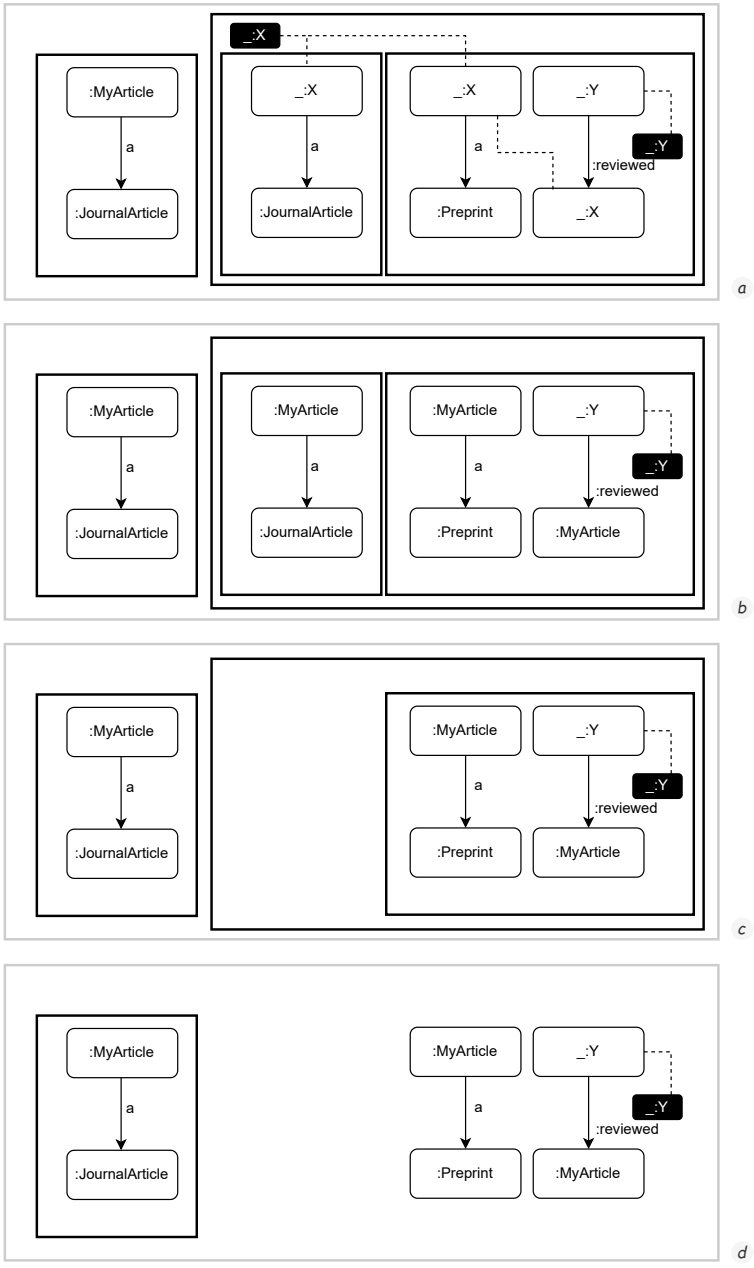


Fig. 12. (a) Listing 1.5 with the added ¬⟨`:MyArticle a :JournalArticle`⟩ on the default (positive) surface; (b) deiteration of graffiti nodes by setting `_:X` equal to `:MyArticle`; (c) deiteration using rule **R4**; (d) removing the double cut using rule **R3**.

```
1  @prefix log: <http://www.w3.org/2000/10/swap/log#> .
2  @prefix : <https://example.org/ns#> .
3
4  ( _:X ) log:onNegativeSurface {
5
6      _:X a :Publication .
7
8      ( ) log:onNegativeSurface {
9          ( ) log:onNegativeSurface {
10
11             ( ) log:onNegativeSurface {
12                 _:X a :Journal .
13             } .
14
15             ( ) log:onNegativeSurface {
16                 _:X a :Book .
17             } .
18         } .
19      } .
20 } .
```

Listing 1.6. An RDF Surfaces graph with a disjunction in the conclusion with meaning: "Anything that is a publication is a journal or a book, or both."

Thus, complex implications can be composed using a copy-and-paste approach with simpler constructs. The pattern of nested negative surfaces on lines 4-8 starts an implication, and the pattern of negative surfaces on lines 9-18 is that of a disjunction.

6 RDF Surfaces reasoner implementation

While Peirce's diagram rules provide one approach to the logic inferencing of RDF Surfaces, it is not the sole method available. We use Peirce's diagram rules in this paper to convince the reader of the application of FOL using RDF Surfaces. Using four relatively simple rules, authors of RDF Surfaces can make simple derivations and check the consequences of surface logic by hand. However, this does not imply that automated systems must use this derivation method. The world of automatic theorem proving is extensive, featuring many FOL provers and an active community including a "World Championship for Automated Theorem Proving" (CADE).[10]

We are currently experimenting with four implementations specially targeted to RDF Surfaces. Only one implementation uses a direct translation of Peirce diagram rules in its codebase:

- *EYE* [14] is implemented in SWI-Prolog[11] and is based on forward and backward chaining. The resolution algorithm rewrites RDF Surfaces into a conjunction of disjunctive normal forms (DNF). The codebase was originally written to support the syntax and semantics of Notation3 but also implements RDF Surfaces semantics.
- *Retina* [15] is also implemented in Prolog and can be run in Trealla[12] and Scryer[13]. The codebase is a rewrite of EYE targeted to processing RDF Surfaces.
- *Latar* [28] uses a calculus that is directory inspired by Beta graph reasoning implemented in SWI-Prolog.
- *Tension.js* [61] is implemented in Typescript and uses a similar resolution algorithm as EYE.

Of all these implementations, EYE is the most mature and was used extensively in our experiments to apply RDF Surfaces to real-world use cases. EYE provides a command line and a browser version. At https://w3c-cg.github.io/rdfsurfaces/demonstrator/, an experimental RDF Surfaces implementation using EYE is available. The following RDF Surfaces features are already available in EYE:

- Full support of the RDF Surfaces syntax as presented in this paper.
- Explicit scoping of logical variables.

[10] https://tptp.org/CASC/
[11] https://www.swi-prolog.org
[12] https://github.com/trealla-prolog/trealla
[13] https://www.scryer.pl

- Existential closure of free variables on the default (positive) surface.
- Explicit negation of triples and conjunctions of triples using the `log:onNegativeSurface` predicate.
- Disjunctions in the data, antecedent and consequent of an implementation.

In the current version, EYE v10.10.0, there are some known limitations. Due to the extensive support for forward and backward chaining, our focus was mainly on use cases with an implication structure. Additionally, every implementation is expected to have computational limits due to the undecidable nature of the underlying logic. It is possible to create formulas that never halt. Creating formulas that will result in an incomplete answer is possible.

EYE, building on its foundations in Notation3 reasoning, also offers several extensions that are not part of the core RDF Surfaces, including functional predicates for list, string, and mathematical calculations.

In EYE and all other RDF Surfaces implementations, there is no distinction between assertion and theory boxes, which are typically required in many knowledge processing systems. RDF Surfaces transport data and logic using a common embedded RDF syntax; all assertions and theory statements can be combined in one document.

Two methods are available for querying data in an RDF Surfaces graph in all implementations.

Proof by contradiction checks if a graph G is available in the knowledge base by adding the negated $\neg G$ to the knowledge base and tests if this leads to a contradiction.

Proof by negation checks if a graph $\neg G$ is available in the knowledge base by adding negated $\neg\neg G \equiv G$ to the knowledge base and tests if this leads to a contradiction.

A contradiction in EYE is implemented by an "inference fuse." The EYE reasoner will stop running when such an "inference fuse" is detected and will display the context in which it happened.

Both proof methods are based on classical logic but are quite limited. They only provide yes/no answers: is a graph part of a knowledge base (and its derivations) or not? To answer a more generic query, "show me all triples and derived triples that match a triple/graph pattern," an experimental new surface was added to EYE: the query surface `log:onQuerySurface`. To each RDF Surfaces document, one or more query surfaces can be appended to inspect which bindings for graffiti nodes are available. Listing 1.7 provides an example of a query surface that asks which patterns can be found in a knowledge base for accreditations.

```
1 (_:S _:O) log:onQuerySurface {
2     _:S :accredit _:O .
3 } .
```

Listing 1.7. An example query surface to retrieve all triples and derived triples that match a triple pattern.

Applied to Listing 1.1 this query surface should result in the following triples on the standard output:

```
_:e1 :accredit :JournalA .
```

Proof by contradiction and proof by negation are common derivation techniques. However, contradictions can appear in RDF Surfaces without adding a negated query. The Web can and will be contradictory. Web agents require an approach to address this challenge. Due to the principle of explosion, anything can be proven from a contradiction. If a knowledge base leads to a contradiction, something must be wrong for bad and good reasons. A bad reason includes, for instance, errors in a knowledge base that require correction or data from unreliable sources that should be disregarded. A good reason could include genuine conflicts in human knowledge[14], which indicate starting points for research and academic discourse. In both cases, human intervention or heuristics based on some oracle could decide which data to include in a derivation and how to resolve conflicts. In our opinion, these explicit contradictions are safer on an open Web – in some way even desired – rather than implicit assumptions about negative information with possible contradictions that cannot be discovered.

7 Examples

With the help of the results of Section 5 and Section 4, we are now ready to apply them to the use cases presented in Section 2.

7.1 Scholarly communication

For the scholarly communication use case, we envision a scholarly network of researchers who share their preferences when searching for publication venues. These preferences are preference documents that are serialized as N3S.

[14] These days, we could add machine-generated knowledge when addressing this issue.

These policies can contain information on what preference is regarded as positive, what certainly is not the case or preferences that are only "true" under some conditions. The latter takes the form of an implication. Section 2.1 provides an example of such a researcher preference:

– *Researcher X preferences*: Researcher X prefers preprints in a subject repository, a journal that does not charge APC costs, or a journal indexed in WOS.

One way to express this preference in FOL is by using disjunctions in the antecedent of an implication, as demonstrated in Equation (8).

$$\begin{aligned}\forall \mathtt{x}: \Big(& \langle \mathtt{x\ a\ :SubjectRepository} \rangle \vee \\ & (\langle \mathtt{x\ a\ :Journal} \rangle \wedge \neg \langle \mathtt{x\ :charges\ :APC} \rangle) \vee \\ & (\langle \mathtt{x\ a\ :Journal} \rangle \wedge \langle \mathtt{:WOS\ :indexed\ x} \rangle) \Big) \\ & \longrightarrow \langle \mathtt{x\ a\ :ResearcherPreference} \rangle .\end{aligned} \tag{8}$$

For the preference Equation (8), the universal quantified variable x stands for a publication venue. When the conditions hold for x, it is a preference for a particular researcher.

An alternative way to formulate this preference with the same meaning can be seen in Equation (9).

$$\begin{aligned}& \Big(\forall \mathtt{x} : \langle \mathtt{x\ a\ :SubjectRepository} \rangle \\ \rightarrow & \langle \mathtt{x\ a\ :ResearcherPreference} \rangle \Big) \wedge \\ & \Big(\forall \mathtt{x} : (\langle \mathtt{x\ a\ :Journal} \rangle \wedge \neg \langle \mathtt{x\ :charges\ :APC} \rangle) \\ \rightarrow & \langle \mathtt{x\ a\ :ResearcherPreference} \rangle \Big) \wedge \\ & \Big(\forall \mathtt{x} : (\langle \mathtt{x\ a\ :Journal} \rangle \wedge \langle \mathtt{:WOS\ :indexed\ x} \rangle) \\ \rightarrow & \langle \mathtt{x\ a\ :ResearcherPreference} \rangle \Big) .\end{aligned} \tag{9}$$

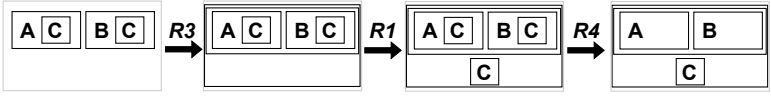
We made here use of the fact that $(A \rightarrow C) \wedge (B \rightarrow C) \equiv (A \vee B) \rightarrow C$ which can be easily derived using Peirce's Alpha system[15]. This alternative formulation has the advantage that formula Equation (9) as an RDF Surfaces serialization does not require the double nesting of all disjunction elements. New preference options can be added to the RDF Surfaces preferences document by appending a new implication pattern rather than inserting a new RDF Surfaces statement into an existing disjunction. The translation of the preferences of researcher X, as stated in Equation (9), into RDF Surfaces is provided in Listing 1.8.

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

# Pref 1 . Publications in a subject repo
(_:X) log:onNegativeSurface {
  _:X a :SubjectRepository .

  () log:onNegativeSurface {
    _:X a :ResearcherPreference.
```

[15] The derivation starts on the left with the diagram for $(A \rightarrow C) \wedge (B \rightarrow C)$. Using diagram rule **R3**, a double negated surface can be drawn around this diagram. Using diagram rule **R1**, a new C symbol can be added on the outer negative surfaces with parity 1. With diagram rule **R4** this C can be deiterated from the inner nested surfaces with nesting level 3. The result is the diagram for $(A \vee B) \rightarrow C$.



```
  } .
} .

# Pref 2 . Publications by a journal that doesn't charge APC costs
(_:X) log:onNegativeSurface {
    _:X a :Journal .

    () log:onNegativeSurface {
        _:X :charges :APC .
    } .

    () log:onNegativeSurface {
        _:X a :ResearcherPreference.
    } .
} .

# Pref 3 . Publications by a publisher that is in WOS
(_:X) log:onNegativeSurface {
   _:X a :Journal .
   :WOS :indexed _:X .

   () log:onNegativeSurface {
       _:X a :ResearcherPreference.
   } .
} .
```

Listing 1.8. The translation of the symbolic Equation (9) into RDF Surfaces N3S.

The same procedure can be done for the departmental Y preferences that were stated as follows:

– *Department Y preferences*: All publications must be journals that are indexed in WOS.

A direct translation into FOL follows in Equation (10).

$$\forall \mathtt{x} : (\langle \mathtt{x\ a\ :Journal} \rangle \wedge \langle \mathtt{:WOS\ :indexed\ x} \rangle) \rightarrow \langle \mathtt{x\ a\ :DepartmentPreference} \rangle) \tag{10}$$

The RDF Surfaces version of Equation (10) is available Listing 1.9.

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

# Pref 1 . Only journals that are indexed in WOS
(_:X) log:onNegativeSurface {
   _:X a :Journal .
   :WOS :indexed _:X .

   () log:onNegativeSurface {
      _:X a :DepartmentPreference .
   } .
} .
```

Listing 1.9. The translation of the symbolic Equation (10) into RDF Surfaces N3S.

The policies for publishing venues can be spread around the Web. Sources that describe journal information can include information such as the journal homepage, the publisher's website, the library database, and indexing services, such as Web of Science, to name a few. The nature of this information is inherently decentralized and, when provided as Linked Data, distributed using many ontologies. Establishing a common method for creating negative facts becomes essential in such an environment. For instance, it is quite common for journals to charge APC costs. That a particular journal does not charge APC costs is a negative fact that benefits the budgets of many researchers. In Equation (11), we symbolize a summary of publishing venue facts that state:

– ABC, DEF, and GHI are journals.

– JKL is a subject repository.
– ABC charges APC costs, but GHI does not charge these costs.
– ABC and DEF are indexed in the WOS database, but GHI is not.

$$\begin{aligned}&\langle \texttt{:ABC a :Journal}\rangle \wedge \langle \texttt{:DEF a :Journal}\rangle \wedge \langle \texttt{:GHI a :Journal}\rangle \wedge \\ &\langle \texttt{:JKL a :SubjectRepository}\rangle \wedge \\ &\langle \texttt{:ABC :charges :APC}\rangle \wedge \neg\langle \texttt{:GHI :charges :APC}\rangle \wedge \\ &\langle \texttt{:WOS :indexed :ABC}\rangle \wedge \langle \texttt{:WOS :indexed :DEF}\rangle \wedge \neg\langle \texttt{:WOS :indexed :GHI}\rangle\end{aligned} \tag{11}$$

The translation of Equation (11) into RDF Surfaces is provided in Listing 1.10.

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

## Journal facts
:ABC a :Journal .
:DEF a :Journal .
:GHI a :Journal .

# APC facts
:ABC :charges :APC .

## GHI is a journal that does not require APC costs
() log:onNegativeSurface {
    :GHI :charges :APC .
} .

# WOS Facts
:WOS :indexed :ABC , :DEF .

() log:onNegativeSurface {
    :WOS :indexed :GHI .
} .

## JKL is a subject repository
:JKL a :SubjectRepository .
```

Listing 1.10. The translation of the symbolic Equation (11) into RDF Surfaces N3S.

In our vision, facts about publishing venues are publicly shared. This allows Web agents to harvest and match the data against researcher and department policies. Consequently, this process can be used to make informed publishing recommendations as a service. For instance, when a Web agent gets hold of the policies expressed in Equation (8), Equation (9), Equation (10), and Equation (11), a logical query can be posed that asks if journal ABC is a researcher and department preference, by adding the negation of $\langle$`:ABC a :ResearcherPreference`$\rangle \wedge \langle$`:ABC a :DepartmentPreference`$\rangle$ to the knowledge base and test if this leads to a contradiction.

At https://w3c-cg.github.io/rdfsurfaces/demonstrator/, an experimental RDF Surfaces implementation using EYE is available to test the examples provided in this section. The RDF Surfaces from this section can be consulted as one resource at https://w3c-cg.github.io/rdfsurfaces/examples/scholary$_p$*ublication.n*3.

For illustration purposes, we assume a Web agent wants to consult these preferences and adds a negated query at the end:

```
() log:onNegativeSurface {
    :ABC a :DepartmentPreference .
    :ABC a :ResearcherPreference .
} .
```

The negation:

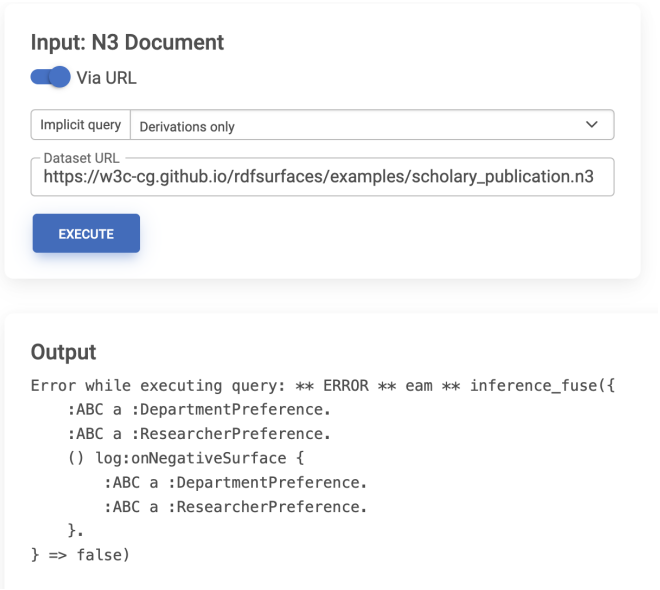


Fig. 13. If adding negated triples to an RDF Surfaces document results in a contradiction (expressed as an inference fuse in EYE), the positive version of the triples can be asserted instead. In this example ⟨`:ABC a :ResearcherPreference`⟩ and ⟨`:ABC a :DepartmentPreference`⟩ is part of the deductive closure.

$$\neg(\langle \texttt{:ABC a :ResearcherPreference} \rangle \wedge \langle \texttt{:ABC a :DepartmentPreference} \rangle)$$

leads to a contradiction. Therefore, the Web agent can conclude that journal ABC is a researcher and department preference.

Figure 13 illustrates this example where, after specifying the location of the RDF Surfaces knowledge base plus the negated query, the derivation leads to an "inference fuse", indicating a contradiction.

7.2 Medicine Prescription

In the healthcare domain, we envision knowledge systems that include both positive and negative information about medications and policies on how these medications can be prescribed to patients with pre-existing conditions. These systems would consider that for example certain medications may cause allergic reactions when prescribed to patients. Consider a collection of medicines for a medical prescription use case: high-dosage aspirin, low-dosage aspirin, and beta-blockers. A high dosage of aspirin is an effective treatment for a fever. In turn, a low dosage of aspirin or a prescription of beta-blockers are both considered to be an effective treatment for acute myocardial infarction. However, both high and low dosages of aspirin may only be prescribed when a patient is known not to be allergic to aspirin. Aspirin can also not be prescribed when a patient has active peptic ulcer disease. Beta-blockers, on the other hand, cannot be prescribed when a patient suffers from severe asthma.

We assume the following expert policies in terms of medicine prescription:

- A low aspirin dosage can be prescribed for a patient with a fever and with neither an allergy to aspirin nor active peptic ulcer disease.
- For a patient with acute myocardial infarction but without an allergy to aspirin or active peptic ulcer disease, a high dosage of aspirin can be prescribed.
- For a patient with acute myocardial infarction but without severe asthma or chronic obstructive pulmonary disease, beta-blockers can be prescribed.

The first policy indicates that if a given patient has a fever (A), one of the following must hold: the patient has an aspirin allergy (B), has active peptic ulcer disease (C), or is prescribed a high dosage of aspirin (D). This can be inferred from the logical equivalence $\neg(A \wedge \neg B \wedge \neg C \wedge \neg D) \equiv (A \wedge \neg B \wedge \neg C) \rightarrow D \equiv A \rightarrow (B \vee C \vee D)$. In

other words, the Peircian diagram that corresponds with these formulas has equal readings[16]. Any of these three forms could be used to translate the policy in a FOL format. In Equation (12), we use a symbolic form that closely matches the method applied in Equation (9) for researcher preferences.

$$\begin{aligned}\forall \texttt{x} : \Big(& \langle \texttt{x :has :Fever} \rangle \wedge \\ & \neg \langle \texttt{x :has :AllergyForAspirin} \rangle \wedge \\ & \neg \langle \texttt{x :has :ActivePepticUlcerDisease} \rangle \Big) \\ & \longrightarrow \langle \texttt{x :isPrescribed :aspirinHighDose} \rangle \end{aligned} \tag{12}$$

The translation for the other two pieces of expert knowledge can be constructed analogously. The RDF Surfaces version of these policies is provided in Appendix A.

With the expert knowledge encoded in RDF Surfaces, we are able to infer the correct prescription for a patient, given their medical condition as well as allergies (or lack thereof). We consider the following two patients, Ann and Joe. Ann has a fever and does not have an allergy to aspirin or an active peptic ulcer disease. Joe suffers from acute myocardial infarction and is allergic to aspirin. However, he does not have active peptic ulcer disease, severe asthma, or chronic obstructive pulmonary disease. These facts can be translated into symbolic form as demonstrated by Equation (13) and Equation (14) with a translation in RDF Surfaces in Appendices B and C.

$$\begin{aligned} & \langle \texttt{:Ann :has :Fever} \rangle \wedge \\ & \neg \langle \texttt{:Ann :has :AllergyForAspirin} \rangle \wedge \\ & \neg \langle \texttt{:Ann :has :ActivePepticUlcerDisease} \rangle \end{aligned} \tag{13}$$

$$\begin{aligned} & \langle \texttt{:Joe :has :AcuteMyocardialInfarction} \rangle \wedge \\ & \langle \texttt{:Joe :has :AllergyForAspirin} \rangle \wedge \\ & \neg \langle \texttt{:Joe :has :ActivePepticUlcerDisease} \rangle \wedge \\ & \neg \langle \texttt{:Joe :has :SevereAsthma} \rangle \wedge \\ & \neg \langle \texttt{:Joe :has :ChronicObstructivePulmonaryDisease} \rangle \end{aligned} \tag{14}$$

The RDF Surfaces from Appendices A, B, and C can be consulted as one resource at https://w3c-cg.github.io/rdfsurfaces/exam
doseaspirinforpatientAnnandabetablockerforpatientJoe.Theprocedureissimilartotheoneinthepreviussection.Wecanaddtoth

```
() log:onNegativeSurface {
    :Ann :isPrescribed :aspirinHighDose .
    :Joe :isPrescribed :betaBlocker .
} .
```

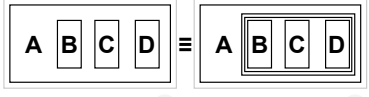
Figure 14 illustrates this example where, after copy and pasting the RDF Surfaces knowledge base and adding the negative query, the derivation leads to an "inference fuse", indicating a contradiction. The *negation*:

$$\neg(\langle \texttt{:Ann :isPrescribed :aspirinHighDose} \rangle \wedge \langle \texttt{:Joe :isPrescribed :betaBlocker} \rangle)$$

leads to a contradiction. Therefore, we can conclude that patient Ann can be prescribed a high aspirin dosage, and patient Joe can be prescribed a beta-blocker.

Are these the only conclusions that can be drawn from this knowledge base and patient data? To know who can be prescribed what medication instead of a negative surface, the EYE query surface can be added to the knowledge base:

[16] Diagram (a) can be read in many ways: $\neg(A \wedge \neg B \wedge \neg C \wedge \neg D)$, $(A \wedge \neg B \wedge \neg C) \rightarrow D$, $(\neg B \wedge \neg C \wedge \not D) \rightarrow \neg A$, $(A \wedge \neg C \wedge \neg D) \rightarrow \neg B$, etc. Diagram (b) can be read as $A \rightarrow (B \vee C \vee D)$ and is diagram (a) with an added double negative surface around the negated B, C, and D which can be removed using diagram rule **R3**.



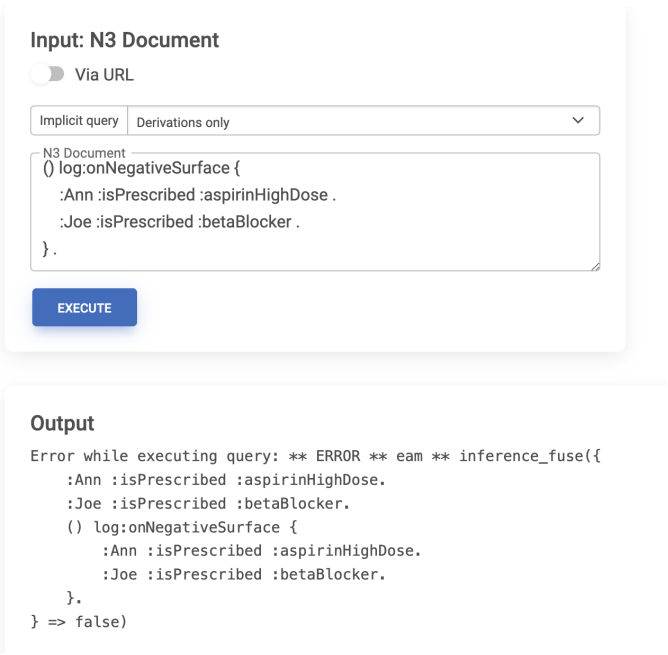


Fig. 14. Addition of a negated conjunction to the healthcare knowledge base plus patient data leads to a contradiction. Therefore, the positive version of the conjunction can be asserted.

```
(_:WHO _:WHAT) log:onQuerySurface {
    _:WHO :isPrescribed _:WHAT .
} .
```

When this query is run in the reasoning interface, as demonstrated in Figure 15, the output will include:

```
@prefix : <https://example.org/ns#>.

:Ann :isPrescribed :aspirinHighDose .
:Joe :isPrescribed :betaBlocker .
```

In this case the reasoner will find any possible binding in the deductive closure where

$$\forall \mathtt{x}, \mathtt{y} : \langle \mathtt{x}\ \mathtt{:isPrescribed}\ \mathtt{y} \rangle$$

leads to a contradiction. The query surfaces only includes RDF triples (not N3S) and in case of a contradiction it provides counter examples following this scheme.

8 Discussion and future road map

This paper discussed the need for an expressive Semantic Web Logic extending RDF. We especially identified classical negation as crucial for many use cases, including scholarly communication and healthcare. Therefore, we proposed RDF Surfaces, provided an abstract and a concrete syntax and first reasoner implementation, and showed how our use cases could benefit from it. The authors are currently experimenting with implementing RDF Surfaces in reasoners, and future experiments using existing FOL reasons from the TPTP project are on the near horizon. In 2022, a W3C Community group was formed to define the semantics and discuss the general requirements for implementations and Web logic.[17,18] Of course, this new logic comes with many open challenges, and this vision paper is only the starting point of a longer endeavor. Below, we list the most important challenges ahead and briefly discuss possible ways to solve them.

[17] https://www.w3.org/community/rdfsurfaces/
[18] https://w3c-cg.github.io/rdfsurfaces/

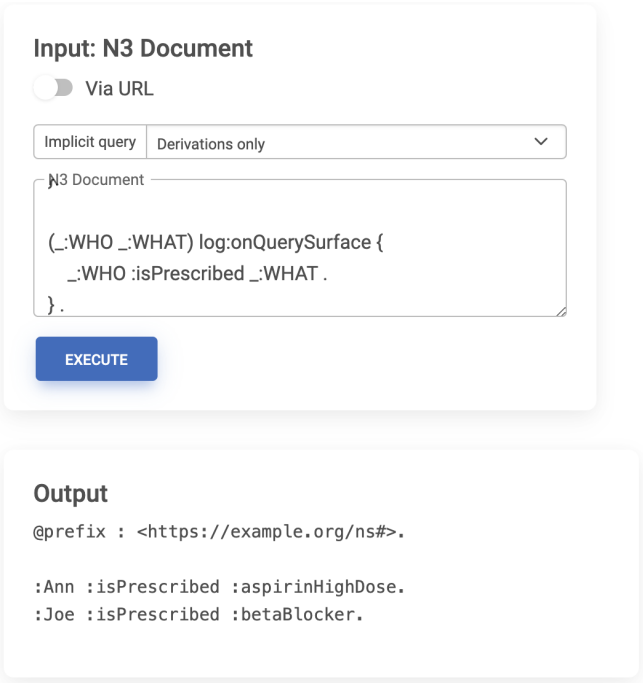


Fig. 15. The addition of a query surface to an RDF Surfaces document can be used to bind all graffiti nodes and inspect all triples that can be derived.

8.1 Negation on the Web

With RDF Surfaces, we provided logic and syntax to express negative information on the Web using RDF. To implement negation, we extended the RDF model with surfaces that describe a (possibly empty) set of negated triples. These triples are quantified by adding graffiti nodes that act as existentially quantified variables. Combining these features with the default conjunction of triples under simple entailment semantics provides the full expressivity of FOL in RDF. Ambiguities in the quantification can be avoided by providing an explicit scope for graffiti nodes placed "on" a surface. We demonstrated how to solve real-world use cases in scholarly communication and healthcare that contain shared negative information, disjunctions, and implications in a decentralized setting. Using RDF Surfaces, Hayes's BLOGIC vision can be realized in concrete implementations, such as the EYE reasoner demonstrated in this paper.

The expressivity and portability of FOL provide the advantage that we can state what we want to say instead of only stating what machines can process. The Semantic Web is an endeavor to express the combined human knowledge irrespective of the computability of this knowledge. Consequently, we do not anticipate that any realistic Web reasoner will be capable of delivering a fully portable solution (P) and accepting every conceivable FOL input, consistently providing complete (C) and exhaustive responses to any query, and invariably terminating (H) within a finite time frame. Demanding these (P)+(C)+(H) attributes simultaneously is infeasible due to the undecidability of FOL and forms an iron triangle: at most, only two of these features can be selected for any implementation. If we want a portable Web logic, feature (P) is unavoidable. Any portable Web reasoner must choose to be complete (C) or always halt (H) in a finite time. In practice, this will lead to a necessary fragmentation and decentralization of the Web where many Web agents must compete to find inconsistencies or find all derivations that can be made from a decentralized knowledge graph.

When a derivation or inconsistency is found, verifying that the given solution can be derived from the knowledge graph is possible. As Trakhtenbrot [58] demonstrated, this is a semi-decidable procedure[19]. A more interesting solution would be publishing proof of every derivation. Rather than publishing what is a (derived) fact on the Web, Web agents could publish the proof of how they concluded these facts. For instance, in a scholarly communication setting, one would like to know why a journal was selected as a valid preference for a researcher. In a healthcare setting, one would trace why or why not a medication was provided to a patient.

[19] A semi-decidable procedure always halts and says "yes" when a statement S validly derives a knowledge base K but might not halt if S is an invalid derivation.

We do not anticipate that every Web author should use FOL terminology to express knowledge and create logical formulas. We regard RDF Surfaces as a low-level language that can transport data and logic to the Web. Higher level vocabularies are still required to create a compact serialization for human consumption (human editors). The pragmatic choice will be to exchange knowledge using higher level vocabularies that can be "compiled" into RDF Surfaces logic. One such example is from the Flemish SHARCS project[20] where a demonstrator was created to make ODRL policy actionable by compilation them to RDF Surfaces.[21] A similar approach using RDF Surfaces was taken by Zhuo and Zhuo (2024).

Extensions to RDF Surfaces would certainly be helpful when expressing non-monotonic negation. For instance, in the scholarly communication use case, expressing which topics are not part of a database is not always feasible. Notation3 provides the capabilities for scoped negation as failure that might need to be combined with RDF Surfaces. Real-world use cases also need to provide input on which extended predicates are required for things that are not easily expressable in FOL, such as calculations, string manipulation, and list manipulation. This requires a delicate balance between the expressiveness of the language for general computation use cases and the formal semantics of classic FOL.

8.2 Syntax

A first attempt at a hosting language for RDF Surfaces led us to use the Notation3 syntax in the form of "RDF Surfaces in N3" (N3S), which only requires a reduced set of features. Notation3 was only used for its syntactical features – graph terms for creating a scope around negated triples and list terms for declaring graffiti nodes – but not its semantic features. Some authors of this paper are active participants in both the Notation3 and RDF Surfaces groups, where overlap and differences between both approaches are debated. One point of argument is the scoping of blank nodes that in Notation3 semantics is local to a graph term but in RDF Surfaces explicit within a negative surface. Another point is Patel-Schneider's argument that a same-syntax extension of RDF to FOL necessarily leads to paradoxes due to self-referential statements [49]. We think we can 'wiggle out' self-referential statements due to introducing negative surfaces with a N3 list term in the subject position (where the graffiti nodes are declared). Using N3 list terms, negative surfaces inside an RDF Surfaces document do not have an identifier. Surfaces can be compared and be isomorphic but cannot be referred to. But this does not absolve us from paradoxes because these are very hard to avoid in highly expressive languages. At least, RDF Surfaces does not rely on potential self-referential statements.

8.3 Formal semantics

As RDF Surfaces is introduced as a logic, the obvious next step is to define it formally. We have already provided the syntax in this paper, but the semantics are only discussed informally and must be specified further. There are several possibilities to solve this issue. Given that Pat Hayes' original idea of a BLOGIC was inspired by existential graphs, we could base RDF Surfaces semantics on Beta graphs and map them to this framework. This would make it easy to define a calculus for RDF Surfaces by following existing literature, e.g., Zeman [68] and Dau [12]. A difficulty, however, is that Beta graphs do not have native support for constants. To formalize this, one would need to address this limitation, for example, by mimicking constants using relations in combination with existential quantification. Another possibility would be to base RDF Surfaces semantics on FOL, which is close to the approach we followed in this paper when we explained our examples. An advantage of this approach would be that FOL is well understood regarding expressiveness and limitations. As RDF Surfaces extends RDF, RDF formalizations based on Beta graphs or FOL would have the burden of proof that the semantics is in line with the one of RDF. In the case of a FOL formalization, this could be done following the work of De Bruijn et al. [13]. This last aspect – the fact that RDF Surfaces aim to extend RDF – makes a formalization extending RDF semantics [27] an interesting choice. It would make sense only to consider simple entailment and D-entailment as the semantics or RDFS could be modeled through RDF Surfaces axioms. Of course, these axioms would need to be carefully designed, and the equivalence would need to be proven.

8.4 Implementations

To be of use for the Web, RDF Surfaces should not only be a theoretical framework, but should be applicable in practice. We need implementations that can check and exchange each other's findings to achieve this. We thus need

[20] https://www.imec-int.com/en/research-portfolio/sharcs
[21] https://github.com/eyereasoner/Notation3-By-Example/tree/main/examples/sharcs-odrl

to agree on the explicit syntax, and the N3S-based framework we introduced in this paper could be a starting point for that. Reasoners should use this as input and apply a calculus that is proven correct and – if possible – also complete according to the semantics. Currently, there are four experimental implementations based on our N3S-based syntax: Latar [28], whose calculus is directly inspired by Beta graph reasoning, EYE [14], which is originally a Notation3 reasoner but lately also provides support for RDF Surfaces, retina [15] based on a rewrite of EYE for RDF Surfaces in Trealla and Scryer Prolog, and Tension.js [61] a Typescript implementation. The calculus applied is close to FOL reasoning. To ensure interoperability between these reasoners and encourage more implementors to support RDF Surfaces, we need shared test cases that express the expected behavior of reasoning systems. A first repository providing such test cases is already available[22]. Still, the test infrastructure needs to become more fine-grained to check for different types of problems like simple parsing errors or the correct application of single inference steps. Use cases like the ones we presented above will help generate more test cases and better understand the specific needs the reasoning should satisfy. They could help to decide on optimizations for data storage and inferencing.

8.5 Relation to other Web logics and standards

Alongside the formalization of the logic, its relationship to existing frameworks should be investigated. Since RDF Surfaces supports classical negation, existential quantification, and conjunction, it is likely that it can express FOL, though this claim must be proven. It is particularly interesting to explore how we can express OWL DL and its profiles [45] in RDF Surfaces, but other common Web frameworks should also be considered.

The relationship to RDF Surfaces is clear for RDF and simple entailment, as the latter is defined as an extension of the former. However, the relationship is less clear for RDFS or rule formats like Notation3 Logic [67] or SWRL [30]. And it becomes even more difficult regarding WC3 recommendations like SHACL [36] or SPARQL [21], which both support non-monotonic features. In its current form, as presented in this paper, RDF Surfaces cannot cope with non-monotonicity. Still, other aspects, like querying with recursive property paths, can be covered in RDF Surfaces.

8.6 Extensions

The last aspect mentioned, the current inability to support non-monotonic behavior, illustrates another direction for future research: it could be possible and maybe even necessary to support concepts, such as negation as failure or closed predicates [46], to be suitable to fully support use cases like the ones introduced in this paper. Notation3 and SPARQL come with built-in or filter predicates, which makes it easy to, e.g. perform operations on strings or sum up two integers. As RDF Surfaces is introduced as a logic facilitating practical use cases, it would be beneficial to include such predicates. As a third, but – given the variety of use cases in the Web – certainly not least extension, we believe that support for unasserted triples like it is proposed in RDF-star [23] and unasserted graphs as they exist in N3 would be beneficial for many use cases, like for example the exchange of provenance knowledge and the reasoning about it. To also support the latter, this support for unasserted knowledge should come with a mechanism that could assert it in the case of provenance, for example, if we decide to trust a source.

Of course, this section only provides a few examples of future development, and we hope our readers already have their own visions here. In that sense, we are very curious to see what the future brings, and we plan to work on these and other research to further RDF actively.

9 Conclusion

RDF Surfaces is a concrete implementation of Pat Hayes' BLOGIC vision of portable Web logic based on FOL. In this vision paper, we demonstrated how BLOGIC has its foundations in Peirce's existential graphs and provided a gentle introduction to the calculus of diagrammatic logic. Our contribution provides the translation of BLOGIC into a concrete RDF syntax, a semantic with FOL expressivity and multiple implementations. RDF Surfaces, with its N3S syntax, deviates from Hayes' original "annotated Turtle" and is a sub-language of Notation3 with a reduced set of features: N3 list terms to represent a set of graffiti blank nodes and graph terms to represent a (possibly empty) set of triples. Using these extensions to RDF and adding a surface type as a predicate, the full expressivity of FOL is available using simple entailment. The applicability of RDF Surfaces was demonstrated in two use cases, one from scholarly communication and one from healthcare.

The benefits of RDF Surfaces as Web logic becomes evident in decentralized networks that need to exchange data and the logic behind the data. In our scholarly communication use-case, no facts were published by the researcher

[22] https://github.com/eyereasoner/rdfsurfaces-tests

and institute, but conditions on future facts in the form of preferences. In the healthcare use case, the conditions on which medication would be applicable for (future) a patient were exchanged. In a decentralized network, Web agents should agree on a Web logic to analyze these preferences and conditions for possible triples that can become facts.

Another use case that could benefit from RDF Surfaces is rights policies that need to exchange information about the permissions and prohibitions to access and use information. With RDF Surfaces, policy documents could not only make this information machine actionable but, in addition, in a decentralized setting, contradictions can be discovered between policies at the authoring time (and not at run time). The computer can say "no" to a policy author who does not have to wait for "the customer at the door" to find out something is wrong.

Publishing negative information is part of human activity; it is a part of commercial activity. The soft drink industry's "This drink contains no sugar" marketing statement is an explicit negative fact. For a consumer, this explicit negative information could be the reason to buy this drink. One option could be to add many negative predicates or classes to Web ontologies to publish this negative information. We believe this would obfuscate that classical negation is intended but not expressed and only adds to Hayes' "diamond of confusion."

Computation complexity is an issue, and as Web logic can be used for different purposes, many levels of complexity can be imagined. Adding the undecidability of FOL queries can potentially require vast computer resources or even run forever without providing an answer. Using Sowa argumentation [55], we do not assume that worst-case scenarios will be the norm on the Web. The theorems of Whitehead and Russell's *Principia Mathematica* could already be proven in the 1960s by vacuum-tube machines. The Web runs today with a potential halting problem waiting in every HTML + JavaScript page. Database queries use the SQL language, which has FOL expressivity and can be solved in polynomial time. We advocate with RDF Surfaces to choose for sharing information and the logic behind this information using an RDF syntax, which makes as few assumptions as possible to create FOL expressivity. With FOL, we have semantics that is well understood.

Our research is still young and requires follow-up research to strengthen our claims. Formal semantics needs to be written. Implementations must demonstrate that values can be added to the Semantic Web using FOL expressivity for real-world use cases. Machines must cooperate and demonstrate, using proofs, why particular derivations were made and their reasoning for providing a conclusion. As Web citizens, we can clearly state what is and what is not using RDF Surfaces.

10 Acknowledgements

This work is partly funded by the Andrew W. Mellon Foundation (grant number: 1903-06675), SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10), and the FWO Project FRACTION (Nr. G086822N). The authors would like to thank the W3C RDF Surfaces community group for joint discussions on the requirements for Web logic. The authors would also like to thank Jesse Wright and Ieben Smesseart for creating a Web version of the EYE reasoner and RDF Surfaces as a Web application.

References

1. S. Abiteboul, R. Hull and V. Vianu, *Foundations of databases*, Vol. 8, Addison-Wesley Reading, 1995.
2. R. Angles and C. Gutierrez, Negation in SPARQL, *arXiv preprint arXiv:1603.06053* (2016).
3. D. Arndt, T. Schrijvers, J. De Roo and R. Verborgh, Implicit quantification made explicit: How to interpret blank nodes and universal variables in Notation3 Logic, *Journal of Web Semantics* **58** (2019), 100501. doi:10.1016/j.websem.2019.04.001.
4. S. Battle, A. Bernstein, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIIraith, D. McGuiness, J. Su and S. Tabet, Semantic Web Services Language (SWSL), 2005.
5. S. Bechhofer, F. Van Harmelen, I. Horrocks, D. McGuiness, P.F. Patel-Schneider and L.A. Stein, OWL Web Ontology Language Reference.
6. D. Beckett, T. Berners-Lee, E. Prud'hommeaux and G. Carothers, RDF 1.1 Turtle, 2014.
7. T. Berners-Lee, D. Karger, L.A. Stein, R. Swick and D. Weitzner, SWELL - Semantic Web Logic Language, 2000.
8. T. Berners-Lee, D. Connolly, L. Kagal, Y. Scharf and J. Hendler, N3Logic: A logical framework for the World Wide Web, *Theory and Practice of Logic Programming* **8**(3) (2008), 249–269. doi:10.1017/S1471068407003213.
9. C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler and M. Smith, OWL 2 Web Ontology Language, 2012, http://www.w3.org/TR/owl2-syntax/.
10. J. Bollen and H. Van de Sompel, An architecture for the aggregation and analysis of scholarly usage data, in: *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '06, Association for Computing Machinery, 2006, pp. 298–307. ISBN ISBN 978-1-59593-354-6. doi:10.1145/1141753.1141821.

11. D. Brickley and R.V.G. Guha, RDF Schema 1.1, 2014.
12. F. Dau, *The logic system of concept graphs with negation: And its relationship to predicate logic*, Vol. 2892, Springer Science & Business Media, 2003. doi:10.1007/b94030.
13. J. de Bruijn and S. Heymans, Logical Foundations of (e) RDF (S): Complexity and Reasoning, *Lecture Notes in Computer Science* **4825** (2007), 86. doi:10.1007/978-3-540-76298-0_7.
14. J. De Roo, Euler Yet another proof Engine, GitHub. https://github.com/eyereasoner/eye.
15. J. De Roo, retina, GitHub. https://github.com/KNowledgeOnWebScale/retina.
16. T. Eiter, G. Ianni and T. Krennwallner, *Answer set programming: A primer*, Springer, 2009. doi:10.1007/978-3-642-03754-2_2.
17. T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer and H. Tompits, Combining answer set programming with description logics for the Semantic Web, *Artificial intelligence* **172**(12–13) (2008), 1495–1539. doi:10.1016/j.artint.2008.04.002.
18. B. Esteves, H.J. Pandit and V. Rodríguez-Doncel, ODRL profile for expressing consent through granular access control policies in SOLID, in: *2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, IEEE, 2021, pp. 298–306. doi:10.1109/EuroSPW54576.2021.00038.
19. T. Feder and M.Y. Vardi, Homomorphism closed vs. existential positive, in: *18th Annual IEEE Symposium of Logic in Computer Science, 2003. Proceedings.*, IEEE, 2003, pp. 311–320. doi:10.5555/788023.789065.
20. M. Genesereth and R.E. Fikes, Knowledge Interchange Format Version 3.0, 1992.
21. S. Harris and A. Seaborne, SPARQL 1.1 Query Language, W3C Recommendation, W3C, 2013, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/.
22. O. Hartig, SPARQL for a Web of linked data: semantics and computability, in: *Proceedings of the 9th international conference on The Semantic Web: research and applications*, 2012, pp. 8–23.
23. O. Hartig, P.-A. Champin, G. Kellogg and A. Seaborne, RDF-star and SPARQL-star, 2023, https://w3c.github.io/rdf-star/cg-spec/editors_draft.html.
24. P. Hayes, Re: [EXT] Re: Upper ontologies from phayes@ihmc.us on 2021-01-20 (semantic-web@w3.org from January 2021).
25. P. Hayes, Re: Why must the Web be monotonic? from Pat Hayes on 2001-07-24 (www-rdf-logic@w3.org from July 2001).
26. P. Hayes, BLOGIC. (ISWC 2009 Invited Talk), 2009. https://www.slideshare.net/PatHayes/blogic-iswc-2009-invited-talk.
27. P. Hayes and P.F. Patel-Schneider, RDF 1.1 Semantics, 2014.
28. P. Hochstenbach, Latar, GitHub. https://github.com/phochste/Latar.
29. A. Hogan, M. Arenas, A. Mallea and A. Polleres, Everything you always wanted to know about blank nodes, *Journal of Web Semantics* **27** (2014), 42–69.
30. I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof and M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML (2004).
31. R. Iannella and S. Villata, ODRL Information Model 2.2, 2018.
32. Information technology — Common Logic (CL) — A framework for a family of logic-based languages, Standard, International Organization for Standardization, Geneva, CH, 2018. https://www.iso.org/standard/66249.html.
33. M.G. Kebede, G. Sileno and T. Van Engers, A critical reflection on ODRL, in: *International Workshop on AI Approaches to the Complexity of Legal Systems*, Springer, 2018, pp. 48–61.
34. B. Ketsman and C. Koch, Datalog with Negation and Monotonicity, in: *23rd International Conference on Database Theory (ICDT 2020)*, Schloss-Dagstuhl-Leibniz Zentrum für Informatik, 2020.
35. M. Kifer, Rule Interchange Format: The Framework, in: *Proceedings of the 2nd International Conference on Web Reasoning and Rule Systems*, 2008, pp. 1–11.
36. H. Knublauch and D. Kontokostas, Shapes Constraint Language (SHACL), Recommendation, World Wide Web Consortium (W3C), 2017. https://www.w3.org/TR/shacl/.
37. H. Knublauch, J. Hendler and I. Kingsley, SPIN - SPARQL Inferencing Notation, 2013.
38. M.-R. Koivunen and E. Miller, W3C Semantic Web Activity, 2001.
39. O. Lassila, Web metadata: A matter of semantics, *IEEE Internet Computing* **2**(4) (1998), 30–37.
40. O. Lassila and R. Swick, Resource Description Framework (RDF) Model and Syntax Specification.
41. C. Lefèvre and P. Nicolas, A First Order Forward Chaining Approach for Answer Set Computing, in: *Logic Programming and Nonmonotonic Reasoning*, E. Erdem, F. Lin and T. Schaub, eds, Springer, pp. 196–208. ISBN 978-3-642-04238-6. doi:10.1007/978-3-642-04238-6_18.
42. N. Matosin, E. Frank, M. Engel, J.S. Lum and K.A. Newell, Negativity towards negative results: a discussion of the disconnect between scientific worth and scientific culture **7**(2) (2014), 171–173. doi:10.1242/dmm.015123.
43. J. Mei and H. Boley, Interpreting SWRL Rules in RDF Graphs **151**(2) (2006), 53–69. doi:10.1016/j.entcs.2005.07.036. https://www.sciencedirect.com/science/article/pii/S1571066106003367.
44. B. Motik, P.F. Patel-Schneider and B. Cuenca Grau, OWL 2 Web Ontology Language Direct Semantics (Second Edition).
45. B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue and C. Lutz, OWL 2 Web Ontology Language Profiles (Second Edition), 2012, https://www.w3.org/TR/owl2-profiles/.
46. N. Ngo, M. Ortiz and M. Simkus, Closed predicates in description logics: Results on combined complexity, in: *Fifteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2016. doi:10.5555/3032027.3032056.

47. G. O'Regan, Computability and Decidability, in: *Mathematical Foundations of Software Engineering: A Practical Guide to Essentials*, G. O'Regan, ed., Springer Nature Switzerland, pp. 229–239. ISBN 978-3-031-26212-8. doi:10.1007/978-3-031-26212-8_14.
48. P.F. Patel-Schneider, Building the Semantic Web Tower from RDF Straw, 2005, pp. 546–551.
49. P.F. Patel-Schneider, RDF: Back to the Graph, 2009.
50. D.D. Roberts, *The Existential Graphs of Charles S. Peirce*, Approaches to Semiotics, De Gruyter Mouton, 1973. ISBN ISBN 9789027925237.
51. J.C. Shepherdson, David Maier and David S. Warren. Computing with logic. Logic programming with Prolog. The Benjamin/Cummings Publishing Company, Menlo Park, Calif., etc., 1988, xxi + 535 pp. **56**(4) (1991), 1495–1495. doi:10.2307/2275495.
52. J. Sowa, *Knowledge Representation: Logical, Philosophical and Computational Foundations*, Brooks/Cole, 2000. ISBN ISBN 0-534-94965-7.
53. J. Sowa, Peirce's Tutorial on Existential Graphs, 2011. http://www.jfsowa.com/pubs/egtut.pdf.
54. J. Sowa, Reasoning with diagrams and images: observation and imagination as rules of inference., *Journal of Applied Logics* **5**(5) (2018), 987.
55. J.F. Sowa, Fads and fallacies about logic, *IEEE Intelligent Systems* **22**(2) (2007), 84–87.
56. U. Straccia and G. Casini, A Minimal Deductive System for RDFS with Negative Statements, arXiv, 2022. http://arxiv.org/abs/2202.13750.
57. T. Tammet and G. Sutcliffe, Combining JSON-LD with First Order Logic, in: *2021 IEEE 15th International Conference on Semantic Computing (ICSC)*, pp. 256–261, ISSN: 2325-6516. doi:10.1109/ICSC50631.2021.00051.
58. B. Trakhtenbrot, On Recursive Separability, in: *DOKL AKAD NAUK SSSR*, Vol. 88, 1953, pp. 953–956.
59. D. Tsarkov, A. Riazanov, S. Bechhofer and I. Horrocks, Using Vampire to Reason with OWL, *Lecture notes in Computer Science* **3298** (2004), 471–485. doi:10.1007/978-3-540-30475-3_33.
60. H. Van De Sompel and P. Hochstenbach, Reference Linking in a Hybrid Library Environment: Part 1: Frameworks for Linking **5**(4) (1999). doi:10.1045/april99-van_de_sompel-pt1.
61. J. Van Herwegen, Tension.js, GitHub. https://github.com/joachimvh/tension.js.
62. R. Verborgh, n3. https://www.npmjs.com/package/n3.
63. R. Verborgh and J. De Roo, Drawing conclusions from linked data on the Web: The EYE reasoner, *IEEE Software* **32**(3) (2015), 23–27. doi:10.1109/MS.2015.63.
64. M. Viswanathan, CS498MV - First Order Logic, 2018. https://courses.physics.illinois.edu/cs498mv/fa2018/FirstOrderLogic.pdf.
65. G. Wagner, Web Rules Need Two Kinds of Negation, in: *Principles and Practice of Semantic Web Reasoning*, Vol. 2901, 2003, pp. 33–50. doi:10.1007/978-3-540-24572-8_3.
66. G. Wagner, C.V. Damasio and G. Antoniou, Towards a general Web rule language, *International Journal of Web Engineering and Technology* **2**(2) (2005), 181–206. doi:10.1504/IJWET.2005.008483.
67. W.V. Woensel, D. Arndt, P.-A. Champin, D. Tomaszuk and G. Kellogg, Notation3 Language, 2023, https://w3c.github.io/N3/reports/20230703/.
68. J.J. Zeman, The graphical logic of CS Peirce, PhD thesis, The University of Chicago, 1964.
69. X. Zhang, J. Van den Bussche and F. Picalausa, On the satisfiability problem for SPARQL patterns, *Journal of Artificial Intelligence Research* **56** (2016), 403–428. doi:10.1613/jair.5028.
70. R. Zhao and J. Zhao, Perennial Semantic Data Terms of Use for Decentralized Web, in: *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 2238–2249. doi:10.1145/3589334.3645631.
71. FIPA RDF Content Language Specification, 2001. http://www.fipa.org/specs/fipa00011/XC00011B.html.

Appendix A Healthcare policies as RDF Surfaces in Notation3

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

(_:WHO) log:onNegativeSurface {

    _:WHO :has :AcuteMyocardialInfarction .

    () log:onNegativeSurface {
        _:WHO :has :AllergyForAspirin .
    } .

    () log:onNegativeSurface {
        _:WHO :has :ActivePepticUlcerDisease .
```

```
    } .

    () log:onNegativeSurface {
        _:WHO :isPrescribed :aspirinLowDose .
    } .

} .

(_:WHO) log:onNegativeSurface {

    _:WHO :has :AcuteMyocardialInfarction .

    () log:onNegativeSurface {
        _:WHO :has :SevereAsthma .
    } .

    () log:onNegativeSurface {
        _:WHO :has :ChronicObstructivePulmonaryDisease .
    } .

    () log:onNegativeSurface {
        _:WHO :isPrescribed :betaBlocker .
    } .
} .
```

Appendix B Patient Ann data as RDF Surfaces in Notation3

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

# patient Ann
:Ann :has :Fever .

() log:onNegativeSurface {
    :Ann :has :AllergyForAspirin .
}.

() log:onNegativeSurface {
    :Ann :has :ActivePepticUlcerDisease .
} .
```

Appendix C Patient Joe data as RDF Surfaces in Notation3

```
@prefix : <https://example.org/ns#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .

# patient Joe
:Joe :has :AcuteMyocardialInfarction.
:Joe :has :AllergyForAspirin.

() log:onNegativeSurface {
    :Joe :has :ActivePepticUlcerDisease .
} .
```

```
() log:onNegativeSurface {
    :Joe :has :SevereAsthma .
} .

() log:onNegativeSurface {
    :Joe :has :ChronicObstructivePulmonaryDisease .
} .
```