

Reasoning about Anonymous Resources and Meta Statements on the Semantic Web

Guizhen Yang¹ and Michael Kifer²

¹ Department of Computer Science and Engineering
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000, U.S.A.
gzyang@CSE.Buffalo.EDU

² Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, U.S.A.
kifer@CS.StonyBrook.EDU

Abstract. Anonymous resources and meta statements are two of the more interesting features of RDF — an emerging standard for representing semantic information on the Web. Ironically, when RDF was standardized by W3C over three years ago [24], it came without a semantics. There is now growing understanding that a Semantic Web language without a semantics is an oxymoron, and a number of efforts are directed towards giving RDF a precise semantics [17, 15, 11].

In this paper we propose a simple semantics for anonymous resources and meta statements in F-logic [23] — a frame-based logic language, which is a popular formalism for representing and reasoning about semantic information on the Web [29, 14, 16, 13, 12].

The choice of F-logic (over RDF) as a basis for our semantics is motivated by the fact that F-logic provides a comprehensive solution for the problem of integrating frames, rules, inheritance, and deduction, and it has been shown to provide an effective inference service for RDF [13, 28].

1 Introduction

RDF [24] was proposed as a standard for representing semantic information on the Web. Ironically, the specification of RDF did not formally define a semantics. Fortunately this peculiar situation is currently being rectified by a number of efforts [17, 15, 11].

While much of the RDF syntax is first-order in nature, *reification*, which is involved in expressing *meta statements* like “Tom believes that Alice said that RDF is a good idea”, is not. In fact, when left to its own devices, reification can lead to logical paradoxes. In Section 8 we explain why this problem does not arise in our framework. Related to this is the issue of anonymous resources, which in RDF are invoked to express statements such as “A person, called Ora Lassila, is the creator of RDF.” In this paper, we propose a simple model-theoretic semantics for both of these issues using F-logic [23] as the underlying formalism.

The choice of F-logic is motivated by several considerations. First, it has been a popular vehicle for ontology-based information mediation and representing semantic information on the Web whenever complex inference is required [20, 22, 29, 14, 16, 12]. As such applications grow in complexity, the need for an inferencing service will increase. For instance, [13] envisions F-logic precisely as such a service for RDF. A number of initiatives (*e.g.*, RuleML³) have sprung up to promote the use of rule-based reasoning for processing semantic information on the Web, and F-logic and its derivatives are some of the languages being considered.

Our contention is that a model theory for RDF must be considered as part of a more general framework, because experience shows that semantics developed for a limited language like RDF might not generalize. For instance, rules and inheritance are some of the issues whose subtle influence is not apparent in the restricted setting of RDF [31]. In fact, we argue that the current proposal for the RDF model theory [15] has several weaknesses, such as *non-compositionality* (see Section 6), which might cause problems down the road. We also point out that there are at least two different useful notions of entailment for RDF graphs (see Section 5), but only one is currently reflected in the RDF model theory document [15].

The idea of embedding RDF into a larger theory is, of course, not new. Embedding RDF into F-logic was proposed in [13], and in [17] the same was done for KIF [19]. Both proposals are incomplete, however, as they do not address reification and anonymous resources — the main subject of the present paper. The embedding into F-logic *would have been* complete if F-logic, as described in [23], had support for these two features. We are rectifying this situation in the present work. Our proposed semantics is conceptually very simple⁴ and is inspired by HiLog [10] — a logic language that provides a foundation for tractable higher-order logic programming — and by the treatment of anonymous object identities in \mathcal{F} LORA-2 [30, 33] — a powerful frame-based language for knowledge representation and reasoning, which is based on F-logic, HiLog, and Transaction Logic [4, 5]. By incorporating reification into the F-logic model theory we provide a model theory for reification in RDF and extend it with powerful meta-programming and inferencing capabilities, which F-logic is known for.

Embedding into F-logic also provides an immediate practical benefit. There are already F-logic based systems, such as \mathcal{F} LORA-2 [30, 33] and TRIPLE [28], which support reification and have RDF handling capabilities. In particular, \mathcal{F} LORA-2 implements the proposed semantics and provides full support for frame-based representation, rules, inheritance, meta-programming, database updates, and more — all in a clean logical fashion. It has already been used in a number of projects ranging from data integration in neuroscience [21] to processing semistructured and semantic information on the Web [12].

³ <http://www.ruleml.org>

⁴ While it is simple, it is not obvious, as a number of authors believed that F-logic does not generalize to deal with reification [7, 13, 28].

This paper is organized as follows. Section 2 surveys the necessary background from F-logic and HiLog. Section 3 motivates the proposed extensions to F-logic from the point of view of modeling using RDF. Section 4 formally treats the semantics of the proposed extensions. Sections 6 and 7 discuss the properties of the semantics introduced in Sections 4 and 5 and point out some problems with the current proposal for the RDF model theory [15]. Section 8 discusses the paradoxes that are often lurking in logical theories that support reification and explains why these problems are avoided in our framework. Section 9 concludes the paper.

2 Preliminaries

In this section we review the main ideas behind F-logic [23] and HiLog [10] — the two formalisms that form the basis for our proposed semantics.

2.1 F-logic

F-logic is an extension of classical predicate logic which allows frame-based (or object-oriented) syntax, and has a natural model-theoretic semantics, and sound and complete proof theory.

F-logic uses Prolog *ground* (*i.e.*, variable-free) terms to represent object identities (abbr., oids), *e.g.*, `john` and `father(mary)`. Objects can have functional (single-valued), multivalued, or Boolean attributes, for example:

```
mary[spouse → john, children →→ {alice, nancy}].
mary[children →→ jack].
mary[married].
```

Here `spouse → john` is a *single-valued* attribute specification; it says that `mary` has an attribute `spouse`, whose value is a singleton oid `john`. The specification `children →→ {alice, nancy}` says that `children` is a *multivalued* attribute; its value in the context of the object `mary` is a set that *includes* the oids `alice` and `nancy`. We emphasize “includes” because a set does not need to be specified all at once. For instance, the second fact above says that `mary` has one additional child, `jack`. Note also that we usually omit the braces while specifying a singleton set. The last clause is an example of a Boolean attribute: it states that the value of `married` is true for the object `mary`.

While some attributes of an object can be specified explicitly as facts, other attributes can be defined using inference rules. For instance, we can derive `john[children →→ {alice, nancy, jack}]` with the help of the following inference rule:

$$X[\text{children} \rightarrow\rightarrow \{C\}] :- Y[\text{spouse} \rightarrow X, \text{children} \rightarrow\rightarrow \{C\}].$$

Here we adopt the usual Prolog convention that capitalized symbols denote variables, while symbols beginning with a lowercase letter denote constants.

F-logic objects can also have *methods*, *i.e.*, functions that return a value or a set of values when appropriate arguments are provided. For instance,

`john[grade(cs305,f2002) → 100, courses(f2002) → {cs305, cs306}].`

says that `john` has a single-valued method, `grade`, whose value on the arguments `cs305` and `f2002` is 100, and a multivalued method `courses`, whose value on the argument `f2002` is a set of oids that contains `cs305` and `cs306`. As attributes, methods can also be defined using rules.

In addition, *class memberships* (e.g., `john:student`), *subclass relationships* (e.g., `student::person`), *types* (e.g., `person[name ⇒ string]`), and many other things can also be specified — both statically, as facts, and dynamically, via rules.

In the sequel, we will consider only multivalued attributes and ignore the rest of the features — the results of this paper extend straightforwardly to include class membership, subclass relationship, methods, types, etc.

2.2 HiLog

HiLog was introduced in [9,10] to provide a convenient syntax and tractable model theory to higher-order logic programming. The main highlights of this language are the variables that can range over both function and predicate symbols and a complete elimination of the barrier between predicate formulas and first-order terms. In this way, HiLog provides a natural syntax and semantics for reification: a statement can be a formula and an object at the same time.

We illustrate HiLog through examples. The simplest yet most unusual one is the definition of the standard Prolog meta-predicate `call`:

`call(X) :- X.`

In this example, HiLog does not distinguish between function terms and atomic formulas: the same variable can range over both. Therefore, one can reify statements and reason about them in the same language. For instance, we can state that Bob believes (among other things) that Mary likes RDF as follows:

`believes(bob,likes(mary,rdf)).`

and then state that whatever Bob believes is true:

`X :- believes(bob,X).`

Variables can also range over function symbols, as in `X(Y, a)`. A query of the form `?- p(X), X, X(Y, X)` is well within the boundaries of HiLog. The syntax for HiLog terms also extends that of classical logic. For instance, `g(X)(f(a, X), Y)(b, Y)` is perfectly fine. Of course, such powerful syntax should be used sparingly, but people have found many important uses for these features. For instance, the following simple program defines transitive closure of *any* binary relation. The program defines a higher-order predicate constructor `closure`, which takes a binary predicate as a parameter and yields a first-order predicate:

`closure(Pred)(X,Y) :- Pred(X,Y).`
`closure(Pred)(X,Y) :- Pred(X,Z), closure(Pred)(Z,Y).`

Here `Pred` is a variable that ranges over predicate symbols. When it is bound to a particular symbol, say `parent`, the above program would compute the relation `closure(parent)`, *i.e.*, the ancestor relation. More examples of this kind of programming are found in [10].

When combined with F-logic, HiLog enables powerful meta-programming features [23, 30, 33]. For instance, we can define an attribute, `methods`, whose value for any object is the set of the names of unary and binary single-valued methods defined for that object:

$$\begin{aligned} X[\text{methods} \rightarrow \{M\}] & :- X[M(A) \rightarrow V]. \\ X[\text{methods} \rightarrow \{M\}] & :- X[M(A1,A2) \rightarrow V]. \end{aligned}$$

Since the main focus of this paper is reification and anonymous identity, in the rest of this paper we will consider only the subset of HiLog that enables reification and combine it with F-logic. Indeed, the use of HiLog to enhance meta-programming in F-logic was discussed in [23, 30], but its use for supporting reification has not been considered.

3 Supporting RDF in F-logic

It was argued in [13] that F-logic is a natural formalism to provide semantics and inference service for RDF(S) [24]. However, some important aspects, such as anonymous resources, containers, and reification were left out because the original F-logic [23] did not support them. In this section we illustrate on a number of examples that all these features can be supported by slightly extending the logic with *anonymous ID symbols* and *reified statements*. Formal treatment of this extension is given in Sections 4 and 5.

3.1 RDF Data Model

First, we briefly recall the RDF data model, which can be represented as triples, as a graph, or in XML [24]. These representations have equivalent meaning but here we will mainly use triple syntax and XML to present RDF data (see [24] for more details). An RDF document is a finite set of statements of the form `{predicate, subject, object}` [24], where `predicate` is a *property*, `subject` is a *resource*, and `object` is a resource or a literal. A set of RDF statements can also be viewed as a *directed labeled graph*, where the vertexes are the resources and the literals, and a triple `{p, s, o}` represents an arc from `s` to `o`, labeled with `p`.

A resource describes a real or conceptual entity (*e.g.*, John Doe). Typically, resources are represented as URIs [2]. But they can also be *anonymous* (*e.g.*, someone). For instance, the URI, `http://www.w3.org/TR/REC-rdf-syntax`, represents the abstract concept of RDF itself. URIs are considered as logical constants referring to entities. In this paper when we use triple syntax to present RDF data, we will enclose URIs using a pair of square brackets, *e.g.*, `[http://foo.org/TheBulb]`. For simplicity we will ignore aspects of meaning encoded in particular URI forms and just treat URI references as simple names.

A property is a predicate that specifies a *binary* relationship (*e.g.*, `parent`). In RDF, properties form a subset of resources and are also represented using URIs. Property names must be associated with a schema or vocabulary. Usually we use QName syntax (see XML Namespaces specification [6] for more details) to denote property names, *e.g.*, `rdf:type`, where `rdf` can be defined as an XML namespace prefix referring to `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, the URI of the (conceptual) vocabulary of RDF properties. For simplicity we sometimes omit the namespace prefix and just write the property name in triple syntax, *e.g.*, `[inventor]`. How XML namespaces are resolved is orthogonal to the results of this paper.

Finally, literals are constants in some primitive data types, such as `string` or `number`. The meaning of a literal is principally determined by its character string: it either refers to the value mapped from the string by the associated data type, or the literal itself when no data type is provided. In all of our examples here literals are just character strings. We normally denote literals using a pair of double quotes in triple syntax, *e.g.*, `"Thomas Edison"`.

However, there is slightly a deviation to represent RDF in F-logic. For simplicity in illustration, we will use atomic constants (sometimes enclosed in single quotes) to denote URIs, properties, and literals, *e.g.*, `'http://foo.org/TheBulb'`, `'rdf:type'`, `'Thomas Edison'`. In real implementation the semantics of these names and notations can be resolved by a separate procedure.

3.2 Anonymous Object Identity

Representation of RDF statements with named resources in F-logic is straightforward. For instance, the following sentence

Thomas Edison is the inventor of the bulb (represented by a resource with the URI `http://foo.org/TheBulb`).

can be represented as a triple

`{[inventor], [http://foo.org/TheBulb], "Thomas Edison" }`

where the notation `[ref]` denotes the resource identified by the URI `ref` and a string enclosed in double quotes denotes a literal. In F-logic, the same statement is written like this:

`'http://foo.org/TheBulb'[inventor → 'Thomas Edison']`.

One difficulty arises when we try to translate RDF statements that include *anonymous resources*, *i.e.*, resources that are not named explicitly. This kind of resources are involved in expressing statements such as

Someone, named Thomas Edison, born in 1847, is the inventor of the resource `http://foo.org/TheBulb`.

The intent here is to make a structured resource *without a known object ID* and state that it has two properties, `name` and `born`, with the above values. In RDF, this sentence would be represented using triple syntax as follows:

```
{[name], [X], "Thomas Edison" }
{[born], [X], "1847" }
{[inventor], [http://foo.org/TheBulb], [X]}
```

Here [X] represents an anonymous resource.

Objects with anonymous IDs were not envisioned in the original work on F-logic [23], but were introduced in \mathcal{F} LORA-2 [33, 30] — our implementation of F-logic that extends it with many additional concepts. To represent such objects, \mathcal{F} LORA-2 uses a special symbol, `_#`, called an *unnumbered anonymous ID symbol*, and another countable set of symbols, `_#1`, `_#2`, ..., etc., called *numbered anonymous ID symbols*. The intended meaning (which is formalized in Section 4) is that each occurrence of `_#` denotes a distinct object ID that does not occur anywhere else in the program. All occurrences of the same numbered anonymous ID symbol, e.g. `_#1`, within the same scope are treated as representing the same object ID, but this ID is distinct from any other ID used elsewhere in the program (including the occurrences of `_#1` in a different scope). The notion of scope is formalized in Section 4, but for our current purposes let us assume that the scope of numbered anonymous ID symbols extends over the entire clause (where each clause is terminated with a “.” and comma represents the conjunction). Thus, the above statement can be represented in \mathcal{F} LORA-2 as follows:

```
'http://foo.org/TheBulb'[inventor → _#1],
_#1[name → 'Thomas Edison', born → '1847'].
```

Note that here the two occurrences of `_#1` are within the same clause and thus the same scope. So they refer to the same object. If we want to state that *someone invented the bulb and someone called Thomas Edison was born in 1847*, then we could write

```
'http://foo.org/TheBulb'[inventor → _#],
_#[name → 'Thomas Edison', born → '1847'].
```

Here we use unnumbered anonymous ID symbols and, even though they occur within the same scope, they represent different objects.

Anonymous resources are also frequently used to represent *containers* in RDF. For example, the following sentence

The committee of Fred, Wilma, and Dino approved the resolution.

can be expressed using the *Bag* container of RDF and would be written in the RDF syntax as follows:

```

<rdf:RDF>
  <rdf:Description about="http://xyz.org/resolution" >
    <approvedBy>
      <rdf:Bag>
        <rdf:li resource="http://xyz.org/members/Fred" />
        <rdf:li resource="http://xyz.org/members/Wilma" />
        <rdf:li resource="http://xyz.org/members/Dino" />
      </rdf:Bag>
    </approvedBy>
  </rdf:Description>
</rdf:RDF>

```

In \mathcal{F} LORA-2, the same sentence can be represented as follows:⁵

```

_#1[
  'rdf:type' → 'rdf:Bag',
  'rdf:_1' → 'http://xyz.org/members/Fred',
  'rdf:_2' → 'http://xyz.org/members/Wilma',
  'rdf:_3' → 'http://xyz.org/members/Dino'
],
'http://xyz.org/resolution'[approvedBy → _#1].

```

Again, here the two occurrences of $_#1$ are within the same scope and thus represent the same object. The first occurrence represents a *Bag* object and the second occurrence refers to this object.

3.3 Reified Statements

Reification in RDF is used to make meta statements, *i.e.*, statements about statements. Since statements are formulas, making statements about them means that formulas must be somehow treated as objects. To represent the following statement

Someone named John Doe believes that a person, called Thomas Edison, invented the bulb (resource <http://foo.org/TheBulb>).

using RDF triple syntax one would have to write a rather convoluted expression below:

⁵ F-logic provides other, more natural ways of expressing the same thing, like, for instance, '<http://xyz.org/resolution>' [approvedBy → {'<http://xyz.org/members/Fred>', '<http://xyz.org/members/Wilma>', '<http://xyz.org/members/Dino>'}]. The cumbersome statement above is intended to mimic the structure of the corresponding RDF statement.

```

{[type], [X], [RDF:Statement]}
{[predicate], [X], [inventor]}
{[subject], [X], [http://foo.org/TheBulb]}
{[object], [X], [Y]}
{[name], [Y], "Thomas Edison"}
{[name], [Z], "John Doe"}
{[believes], [Z], [X]}

```

Here a new, anonymous resource X is used as a *referent* to the following reified statement

A person, called Thomas Edison, invented the bulb.

This is expressed by the first four triples, which say that: (1) X represents an RDF statement; (2) its predicate is `inventor`; (3) its subject is the URI `http://foo.org/TheBulb`; and (4) its object is another anonymous object Y . In the fifth triple we say that this latter object has property `name` with the value "Thomas Edison". The sixth statement says that there is an anonymous object Z with property `name` whose value is "John Doe". Finally, the last statement says that object Z has property `believes` with value X — the anonymous resource that represents the reified statement that *a person, called Thomas Edison, invented the bulb*.

In our extension to F-logic this statement can be modeled in the following much more natural way:

```

-#[
  name → 'John Doe',
  believes →
  ('http://foo.org/TheBulb'[inventor → #1], #1[name → 'Thomas Edison'])
].

```

Note that here the formula `'http://foo.org/TheBulb'[inventor → #1]` *itself* is an object — not some other object ID that refers to this formula. We will argue in Section 7 that this syntax and its corresponding semantics is superior to that of the current proposal for RDF model theory [15]. It also permits more interesting reasoning to be easily performed over reified statements (see Section 7).

We should note that the above syntax is *not* the actual syntax of \mathcal{F} LORA-2 [33] and is also quite different from the F-logic syntax as described in [23]. We decided to depart from that syntax in this paper in order to avoid the distraction that would result from the introduction of additional features of F-logic and \mathcal{F} LORA-2, and in order to simplify the formal development of the model theory in Section 4. In fact, the actual syntax of \mathcal{F} LORA-2 is much richer, which allows to write the above and the earlier sentences more succinctly:

```

-#[
  name → 'John Doe',
  believes →
  ${'http://foo.org/TheBulb'[inventor → #1[name → 'Thomas Edison']}]
].

```

That is, in the actual $\mathcal{F}_{\text{LORA-2}}$ syntax, reification is specified using the $\$\{\dots\}$ construct and the statement inside of $\$\{\dots\}$ is a shorthand for the conjunction of two F-logic statements: `'http://foo.org/TheBulb'[inventor \rightarrow $_ \#1$] and $_ \#1$ [name \rightarrow 'Thomas Edison']`. The interested reader is referred to [33] for the details.

In Section 4, we discuss the notions of RDF graph entailment and equivalence and show that there are at least two such notions, both useful, but only one is currently considered by the RDF model theory proposal [15].

4 Formal Syntax and Semantics

In this section we formally define the syntax and semantics of an F-logic language extended with anonymous identity and reification. We will continue to call this extension “F-logic” in order to avoid introducing yet another name.

To simplify the exposition, we focus on a subset of the new F-logic syntax. The only kind of atoms we consider here is in the form of $o[m \rightarrow v]$, which specifies that the object o has a multivalued method, m , that returns some set of objects, which contains v as a member. The symbols o , m , and v are F-logic terms (to be defined below); they represent the ID of an object, a method, and a value of the method, respectively. In a program, these terms can contain variables in which case they would represent a parameterized collection of objects — one object per variable instantiation. This design makes meta-programming in F-logic as natural as querying.

An F-logic language \mathcal{L} consists of a set of *constants*, \mathcal{C} ; a set of *variables*, \mathcal{V} ; an *unnumbered anonymous ID* symbol, $_ \#$; *numbered anonymous ID* symbols, $_ \#1$, $_ \#2$, ... (for each positive integer); *connectives* including \neg , \vee , \wedge , and \leftarrow ; *quantifiers* including \exists and \forall ; and *auxiliary symbols*, such as comma, parentheses, and brackets. In defining the semantics for F-logic programs, we will assume an *a priori* fixed F-logic language \mathcal{L} .

Intuitively, an occurrence of an unnumbered anonymous ID symbol implies a *distinct* object that is different from any object represented by any other term. Moreover, two occurrences of $_ \#$ represent two distinct objects. Numbered anonymous ID symbols are similar, but with a twist. Different occurrences of $_ \#N$ and $_ \#M$, where $N \neq M$, represent distinct objects. However, different occurrences of $_ \#N$ (with the same number N) within *the same scope* refer to the same object. This meaning of “scope” will be made precise later, when we give a formal semantics.

Formally, F-logic *terms* and *atoms* are constructed inductively as follows. The idea of this construction is borrowed from HiLog [9, 10] and is extended to accommodate reification of F-logic atoms (statements).

Definition 1 (Terms and Atoms). *Given an F-logic language \mathcal{L} , the terms and atoms are defined inductively as follows:*

- Any constant $c \in \mathcal{C}$ is a term.
- Any variable $X \in \mathcal{V}$ is a term.

- Unnumbered or numbered ID symbols, $_#_$, $_#_1$, $_#_2$, ..., are terms.
- If t is a term and t_1, \dots, t_n are terms, then $t(t_1, \dots, t_n)$ is a term.
- Any term in any of the above forms is called a HiLog term. Note that expressions like $p(a, p(X)(Y))(W, q)(f(X))$ are HiLog terms according to this definition.
- If o , m , and v are terms, then $o[m \rightarrow v]$ is a term, also called an F-logic term.
- If A_1 and A_2 are terms, then their conjunction, $A_1 \wedge A_2$, is a term.

Definition 2 (Flat Formula).

Reification: Any term is also a formula. In particular, any HiLog atom or F-logic atom is called an atomic flat (HiLog or F-logic) formula.

Composition: If ϕ and ψ are flat formulas, then so are

- $\neg \phi$
- $\phi \vee \psi$ and $\phi \wedge \psi$
- $\phi \leftarrow \psi$, which is defined to be just a shortcut for $\phi \vee \neg \psi$
- $\exists X \phi$ and $\forall X \phi$, where $X \in \mathcal{V}$ is a variable.

Note that both F-logic and predicate terms (or, more precisely, HiLog terms) are atomic formulas in our language. In this way, both relational and object-oriented programming are supported. The last two cases in Definition 1 and the first case in Definition 2 state that atomic formulas as well as their conjunctions are terms and so are first-class objects in the language. In particular, as we shall see, variables can range over such formulas. This makes the syntax higher-order and provides support for reification. However, the semantics needs to be carefully defined so as to stay tractable and first-order in the sense of [9, 10].

Now we will define interpretations, *i.e.*, the semantic structures that give meanings to F-logic formulas and programs. Our definitions follow the standard convention except that we need to take special care of the anonymous ID symbols and reified statements. To this end, we first have to define the domain of interpretations. In classical logic programming, the domain is typically the set of all ground terms in the language, which is called the Herbrand universe. In our case, however, the domain must also include the constants that are used to interpret the anonymous ID symbols. This idea is formalized next.

Definition 3 (Augmented Herbrand Universe). Let \mathcal{L} be an F-logic language, \mathcal{C} be the set of constants in \mathcal{L} , and \mathcal{D} be a countable set of constants that is disjoint from \mathcal{C} . We shall call \mathcal{D} an anonymous domain, since it will be used to interpret the anonymous ID symbols. The augmented Herbrand universe of \mathcal{L} with respect to \mathcal{D} , denoted $\mathcal{HU}(\mathcal{D})$, is the set of all terms (see Definition 1) constructed using the constants in $\mathcal{C} \cup \mathcal{D}$. Note that in constructing the terms for $\mathcal{HU}(\mathcal{D})$, variables and anonymous ID symbols are excluded. Such variable-free terms are called ground.

Definition 4 (Interpretation). Given an F-logic language \mathcal{L} , an interpretation \mathcal{I} is a pair $(\mathcal{D}, \mathcal{S})$, where

- \mathcal{D} is an anonymous domain, *i.e.*, a countable set of constants that is disjoint from \mathcal{C} (the set of all constants in \mathcal{L}).

- \mathcal{S} is a subset of $\mathcal{HU}(\mathcal{D})$, the augmented Herbrand universe of \mathcal{L} with respect to \mathcal{D} . Moreover, \mathcal{S} is allowed to contain atomic formulas only. Intuitively, \mathcal{S} represents “what is true” in \mathcal{I} .⁶

The above definition differs from those used in classical logic programming, HiLog, and the original F-logic in its use of the anonymous domain. One significant impact of this domain is that the Herbrand universes of different models are different when they use different anonymous domains. To be more precise, the ground terms that are constructed using the constants in \mathcal{C} are the same in all Herbrand universes, but the terms that involve the constants from anonymous domains may not be shared. In classical theory of logic programming all interpretations have the same domain — the Herbrand universe. Our definitions reduce to classical ones when there are no anonymous ID symbols and thus no anonymous domains.

Since interpretations can have different domains, there are many ways to compare interpretations. One possibility is by set inclusion. However, it only makes sense for interpretations over the same domain. Another alternative involves domain mapping and domain isomorphism. Formally, we have the following definitions.

Definition 5 (Domain Mapping). *Let \mathcal{D}_1 and \mathcal{D}_2 be two anonymous domains with respect to an F-logic language \mathcal{L} and let \mathcal{C} be the set of constants in \mathcal{L} . Then any function $\tau: \mathcal{D}_1 \rightarrow \mathcal{D}_2$ is called an anonymous domain mapping. Any function $\lambda: \mathcal{D}_1 \rightarrow \mathcal{D}_2 \cup \mathcal{C}$ is called an augmented domain mapping.*

For any $\mathbf{t} \in \mathcal{HU}(\mathcal{D}_1)$, $\tau(\mathbf{t})$ is a term obtained from \mathbf{t} by simultaneously replacing every constant $\mathbf{d} \in \mathcal{D}_1$ with $\tau(\mathbf{d})$ and leaving the constants in \mathcal{C} intact. For any $\mathcal{S} \subseteq \mathcal{HU}(\mathcal{D}_1)$, $\tau(\mathcal{S}) = \{\tau(\mathbf{x}) \mid \mathbf{x} \in \mathcal{S}\}$. $\lambda(\mathbf{t})$ and $\lambda(\mathcal{S})$ are defined similarly.

Definition 6 (Ordering and Isomorphism). *Let \mathcal{D}_1 and \mathcal{D}_2 be two anonymous domains, and $\mathcal{I}_1 = (\mathcal{D}_1, \mathcal{S}_1)$ and $\mathcal{I}_2 = (\mathcal{D}_2, \mathcal{S}_2)$ be two interpretations.*

- We write $\mathcal{I}_1 \preceq \mathcal{I}_2$ iff there is an 1-1 anonymous domain mapping $\tau: \mathcal{D}_1 \rightarrow \mathcal{D}_2$ such that $\tau(\mathcal{S}_1) \subseteq \mathcal{S}_2$. We will write $\mathcal{I}_1 \prec \mathcal{I}_2$ if $\tau(\mathcal{S}_1) \subsetneq \mathcal{S}_2$.
- \mathcal{I}_1 is isomorphic to \mathcal{I}_2 , denoted $\mathcal{I}_1 \simeq \mathcal{I}_2$, iff $\mathcal{I}_1 \preceq \mathcal{I}_2$ and $\mathcal{I}_2 \preceq \mathcal{I}_1$.

Before we can give semantics to formulas that contain anonymous ID symbols, we need to introduce one more notion, the *scoped formulas*. Recall from Section 3 that the intended meaning of a numbered anonymous ID symbol is that two different occurrences of the same symbol within the scope of the same rule denote the same object; otherwise, they potentially refer to different objects. The notion of scoped formulas allows us to extend this idea to more general types of formulas.

⁶ Note that in classical logic programming an interpretation would contain a subset of the *Herbrand base* — a set of atomic formulas (which is distinct from the set of terms that comprises the Herbrand universe). However, in our case (as in HiLog), atomic formulas are reified and thus the Herbrand base and the Herbrand universe are the same.

Definition 7 (Scoped Formula).

- If ϕ is a flat formula, then $\{ \phi \}$ is a scoped formula.
- If ψ and ξ are scoped formulas, then so are $\psi \vee \xi$ and $\psi \wedge \xi$.

Note that our definition of scoped formulas is such that the scoping braces, $\{ \dots \}$, always apply only to the top level conjuncts and disjuncts. For instance, $\{ \neg \#1[\text{loves} \rightarrow \text{mary}] \} \wedge \{ \exists X(\neg \#1[\text{child} \rightarrow X]) \}$ is a valid scoped formula whereas $\{ \neg \#1[\text{loves} \rightarrow \text{mary}] \} \wedge \exists X(\neg \#1[\text{child} \rightarrow X])$ is not. Although we could define even more general scoping rules, including nested scoping, the practical utility of this complication is unclear and we will not introduce it in this paper.

An *F-logic program* is a finite collection of *scoped rules* where all variables are universally quantified. A rule has the following form:

$$\{ \forall(A_1 \wedge \dots \wedge A_m \leftarrow B_1 \wedge \dots \wedge B_n) \}$$

where $m \geq 1, n \geq 0$, A_i ($1 \leq i \leq m$) and B_j ($1 \leq j \leq n$) are atoms, and only the rule head (the A_i 's) is allowed to have anonymous ID symbols. Note that each rule is a scoped formula where the scope is the entire rule. Thus an F-logic program can be thought of as a scoped formula formed by conjoining all the scoped formulas corresponding to the rules.

Before proceeding, we would like to point out that our theory of anonymous identity, which will be introduced below, is not restricted to clauses of the above form. We choose to limit our attention to rules without negation in the body in order to focus on the problem of modeling of blank nodes in RDF. The theory presented herein can be readily extended to F-logic programs with negation and inheritance by combining it with the semantics described in [31, 32].

Following the standard convention, we will omit universal quantifiers in the rules and since the scope is the entire rule we will omit the scoping braces as well. Thus, rules will be simply written as follows:

$$A_1, \dots, A_m \leftarrow B_1, \dots, B_n$$

We will continue to use the convention from Section 2 whereby uppercase names denote variables and lowercase names denote constants. A rule with an empty body is called a *fact*. When writing down the facts, we will omit the implication symbol and simply show the head.

Definition 8 (Skolemization). Let ϕ be a scoped formula and \mathcal{D} be an anonymous domain. A *skolemization* of ϕ with respect to \mathcal{D} , denoted $\Pi_{\mathcal{D}}(\phi)$, is a formula obtained as follows:

- If $\phi = \psi \vee \varphi$ such that ϕ and φ are scoped formulas, then $\Pi_{\mathcal{D}}(\phi) = \Pi_{\mathcal{D}_1}(\psi) \vee \Pi_{\mathcal{D}_2}(\varphi)$, where $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ and $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$.
- If $\phi = \psi \wedge \varphi$ such that ϕ and φ are scoped formulas, then $\Pi_{\mathcal{D}}(\phi) = \Pi_{\mathcal{D}_1}(\psi) \wedge \Pi_{\mathcal{D}_2}(\varphi)$, where $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2$ and $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$.
- If $\phi = \{ \psi \}$, where ψ is a flat formula, then $\Pi_{\mathcal{D}}(\phi) = \text{skolem}(\mathcal{D}, \psi)$, where $\text{skolem}(\mathcal{D}, \psi)$ is defined as follows:

- Different occurrences of the same numbered ID symbol are mapped into the same constant in \mathcal{D} , but different numbered anonymous ID symbols ($_#\mathbf{M}$ and $_#\mathbf{N}$, where $\mathbf{M} \neq \mathbf{N}$) in ψ are mapped to distinct constants in \mathcal{D} .
- Every occurrence of the unnumbered anonymous ID symbol, $_#\$, in ψ is mapped to a distinct constant in \mathcal{D} (i.e., different from the constants chosen for the numbered ID symbols).

Note that, in a flat formula, each occurrence of the *unnumbered* anonymous ID symbol $_#\$ is mapped to a different element of \mathcal{D} , but all occurrences of the same *numbered* anonymous ID symbol, say $_#\mathbf{1}$, are mapped to the same distinct element of \mathcal{D} . Also observe that two occurrences of the same numbered ID symbol, say $_#\mathbf{1}$, under different scopes are mapped to different elements. For instance, consider a scoped formula $\{ \phi \} \wedge \{ \psi \}$. Then occurrences of $_#\mathbf{1}$ in ϕ are going to be skolemized differently from those in ψ . This fact is a consequence of partitioning \mathcal{D} into disjoint \mathcal{D}_1 and \mathcal{D}_2 in the first two cases of Definition 8.

Another way of looking at $\Pi_{\mathcal{D}}(\phi)$ is that it is just like a flat formula, with no scope and no anonymous ID symbols, but some of the symbols in that formula might come from the anonymous domain \mathcal{D} . The following example illustrates skolemization.

Let $\mathcal{D} = \{\mathbf{o1}, \mathbf{o2}, \mathbf{o3}, \mathbf{o4}, \dots\}$ be an anonymous domain and P be the following simple F-logic program:

$$\begin{aligned} _#\[_#\ \rightarrow _#\mathbf{1}], _#\mathbf{1}[\text{name} \rightarrow \text{mary}]. \\ _#\mathbf{1}[\text{name} \rightarrow \text{'Ora Lassila'}], \text{'RDF'}[\text{creator} \rightarrow _#\mathbf{1}]. \end{aligned}$$

Note that according to our definition of F-logic programs, the above program is a shortcut for the following scoped formula:

$$\begin{aligned} \{ _#\[_#\ \rightarrow _#\mathbf{1}] \wedge _#\mathbf{1}[\text{name} \rightarrow \text{mary}] \} \\ \wedge \\ \{ _#\mathbf{1}[\text{name} \rightarrow \text{'OraLassila'}] \wedge \text{'RDF'}[\text{creator} \rightarrow _#\mathbf{1}] \} \end{aligned}$$

If we map the first occurrence of $_#\$ to $\mathbf{o1}$, its second occurrence to $\mathbf{o4}$, both occurrences of $_#\mathbf{1}$ in the first rule to $\mathbf{o2}$, and both occurrences of $_#\mathbf{1}$ in the second rule to $\mathbf{o3}$, then we obtain the following skolemization $\Pi_{\mathcal{D}}(\text{P})$:

$$\begin{aligned} \mathbf{o1}[\mathbf{o4} \rightarrow \mathbf{o2}], \mathbf{o2}[\text{name} \rightarrow \text{mary}]. \\ \mathbf{o3}[\text{name} \rightarrow \text{'Ora Lassila'}], \text{'RDF'}[\text{creator} \rightarrow \mathbf{o3}]. \end{aligned}$$

A different way of mapping anonymous ID symbols to the constants in \mathcal{D} (e.g., where the first occurrence of $_#\$ is mapped to $\mathbf{o4}$ and the second to $\mathbf{o1}$) would lead to a different skolemization of P. Both mappings are valid skolemizations, however, as they satisfy the conditions of Definition 8.

We can now define what it means to be a model of a given scoped formula.

Definition 9 (Model). Let $\mathcal{I} = (\mathcal{D}, \mathcal{S})$ be an interpretation and ϕ be a scoped formula. Then \mathcal{I} is a model of ϕ , denoted $\mathcal{I} \models \phi$, if and only if there is a skolemization, $\psi = \Pi_{\mathcal{D}}(\phi)$, of the formula ϕ such that $\mathcal{S} \models \psi$, where $\mathcal{S} \models \psi$ is defined in the classical sense:

- If ψ is an atom, then $\mathcal{S} \models \psi$ iff $\psi \in \mathcal{S}$.
- If $\psi = \neg \varphi$, then $\mathcal{S} \models \psi$ iff it is not the case that $\mathcal{S} \models \varphi$.
- If $\psi = \varphi \vee \xi$, then $\mathcal{S} \models \psi$ iff either $\mathcal{S} \models \varphi$ or $\mathcal{S} \models \xi$.
- If $\psi = \varphi \wedge \xi$, then $\mathcal{S} \models \psi$ iff $\mathcal{S} \models \varphi$ and $\mathcal{S} \models \xi$.
- If $\psi = \exists X\varphi$, then $\mathcal{S} \models \psi$ iff there is $t \in \mathcal{HU}(\mathcal{D})$ (a term in the augmented Herbrand universe) such that $\mathcal{S} \models \psi[X/t]$, where $\psi[X/t]$ denotes the formula obtained from ψ by substituting t for all free occurrences of the variable X .
- If $\psi = \forall X\varphi$ then $\mathcal{S} \models \psi$ iff for all $t \in \mathcal{HU}(\mathcal{D})$, $\mathcal{S} \models \psi[X/t]$.

Lemma 1. *If $\mathcal{I} \models \phi$ and $\mathcal{I} \simeq \mathcal{J}$, then $\mathcal{J} \models \phi$. Thus, the set of models of a formula is closed under isomorphism.*

Proof. Let $\mathcal{I} = (\mathcal{D}_1, \mathcal{S}_1)$ and $\mathcal{J} = (\mathcal{D}_2, \mathcal{S}_2)$. Since $\mathcal{I} \models \phi$, it follows that there is a skolemization, $\psi_1 = \Pi_{\mathcal{D}_1}(\phi)$, of the formula ϕ such that $\mathcal{I} \models \psi_1$. Because $\mathcal{I} \simeq \mathcal{J}$, there is an 1-1 anonymous domain mapping $\tau : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ such that $\tau(\mathcal{S}_1) = \mathcal{S}_2$. Let $\psi_2 = \tau(\psi_1)$. By structural induction, we can show that if $\mathcal{I} \models \psi_1$, then $\mathcal{J} \models \psi_2$. It can be easily shown using Definition 8 that ψ_2 is a valid skolemization, $\psi_2 = \Pi_{\mathcal{D}_2}(\phi)$, of the formula ϕ . Therefore $\mathcal{J} \models \phi$.

We can now develop a fixpoint model theory analogously to the classical theory of logic programming. This would provide one computational model for the proposed semantics. Let P be an F-logic program. We will show that P has a property similar to the least model property of logic programs. Of course, given that different models of P can have different anonymous domains, there can be no unique “least” model. However, such models are all isomorphic. These notions are made precise in the following definitions and lemmas.

Definition 10 (Initial Model). *Given an F-logic program P and an interpretation \mathcal{I} , \mathcal{I} is an initial model of P , iff*

- \mathcal{I} is a model of P , i.e., $\mathcal{I} \models P$; and
- \mathcal{I} is minimal, i.e., there is no $\mathcal{J} \models P$ such that $\mathcal{J} \prec \mathcal{I}$.

To construct initial models, we will adapt the classical fixpoint theory for Horn programs to the case of F-logic programs with anonymous ID symbols and reified statements. Namely, we will define a program consequence operator whose least fixpoint computes an initial model of a given F-logic program.

Note that the F-logic programs described here can contain reified statements and a reified statement can be a conjunction of atomic formulas. Therefore, such conjunctions of atomic formulas can be deduced. The consequence of inferring a conjunction is that each constituent atomic formula in the conjunction is also deduced and added to the model. Let A be a conjunction of atomic formulas. We will use $flatten(A)$ to denote the set of atomic formulas that appear in A .

Definition 11 (Program Consequence Operator). *Let P be an F-logic program, \mathcal{D} be an anonymous domain, and $\Pi_{\mathcal{D}}(P)$ be a skolemization of P with respect to \mathcal{D} . Similarly to [25], we can define a program consequence operator,*

$\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}$, which maps an interpretation, $\mathcal{I} = (\mathcal{D}, \mathcal{S})$, over \mathcal{D} to another interpretation over \mathcal{D} as follows: $\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}(\mathcal{I}) = \mathcal{J}$, where $\mathcal{J} = (\mathcal{D}, \mathcal{R})$ and \mathcal{R} is the following set of terms:

$$\left\{ \begin{array}{l} \text{A} \left\{ \begin{array}{l} \text{There is a ground instance } A_1, \dots, A_m \leftarrow B_1, \dots, B_n \\ \text{of a rule in } \Pi_{\mathcal{D}}(\mathcal{P}) \text{ such that:} \\ - A \in \text{flatten}(A_i), \text{ for some } 1 \leq i \leq m; \text{ and} \\ - B_j \in \mathcal{S} \text{ for all } B_j, 1 \leq j \leq n. \end{array} \right. \end{array} \right\}$$

Let $\mathcal{I} = (\mathcal{D}, \mathcal{S})$ and $\mathcal{J} = (\mathcal{D}, \mathcal{R})$ be two interpretations. We write $\mathcal{I} \subseteq \mathcal{J}$ iff $\mathcal{S} \subseteq \mathcal{R}$. Thus \subseteq defines a partial order among all interpretations on the same anonymous domain. It follows from the standard results in logic programming [25] that $\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}$ is monotonic and thus it has a *unique* least fixpoint, denoted $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$.

Theorem 1. *Let \mathcal{P} be an F-logic program, \mathcal{D} be an anonymous domain, and $\Pi_{\mathcal{D}}(\mathcal{P})$ be a skolemization of \mathcal{P} . Then $\mathcal{I} = (\mathcal{D}, \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}))$ is an initial model of \mathcal{P} .*

Proof. Similarly to [25], we can show that $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$ is a model of $\Pi_{\mathcal{D}}(\mathcal{P})$. By Definition 9, it then follows that \mathcal{I} is a model of \mathcal{P} . To show that it is an initial model of \mathcal{P} , we need to show that there is no $\mathcal{J} \models \mathcal{P}$ such that $\mathcal{J} \prec \mathcal{I}$.

Let $\mathcal{J} \models \mathcal{P}$, where $\mathcal{J} = (\mathcal{D}_1, \mathcal{S}_1)$. Since $\mathcal{I} \models \mathcal{P}$, there must exist $\Pi_{\mathcal{D}}(\mathcal{P})$ such that $\mathcal{I} \models \Pi_{\mathcal{D}}(\mathcal{P})$. Similarly, there must exist $\Pi_{\mathcal{D}_1}(\mathcal{P})$ such that $\mathcal{J} \models \Pi_{\mathcal{D}_1}(\mathcal{P})$. Next we will show that $\mathcal{I} \preceq \mathcal{J}$ and hence finish the proof.

If the program \mathcal{P} does not contain any anonymous ID terms, then the two programs, $\Pi_{\mathcal{D}}(\mathcal{P})$ and $\Pi_{\mathcal{D}_1}(\mathcal{P})$, must be the same. It follows from the classical results [25] that $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}) \subseteq \mathcal{S}_1$. Thus $\mathcal{I} \preceq \mathcal{J}$.

If \mathcal{P} contains anonymous ID terms, then there is an 1-1 mapping $\tau: \mathcal{D} \rightarrow \mathcal{D}_1$ such that $\tau(\Pi_{\mathcal{D}}(\mathcal{P}))$, the program obtained by applying τ to every term in the program $\Pi_{\mathcal{D}}(\mathcal{P})$, is the same as $\Pi_{\mathcal{D}_1}(\mathcal{P})$. Then by transfinite induction on each iteration of applying the program consequence operator to generate $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$, we can show that $\tau(\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})) \subseteq \mathcal{S}_1$. It follows that $\mathcal{I} \preceq \mathcal{J}$.

Of course, a program can have different initial models, since there can be different anonymous domains and different skolemizations. However, all initial models turn out to be isomorphic and all are least fixpoints of the $\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})}$ operator. This will be seen from the following two corollaries.

Corollary 1. *$\mathcal{I} = (\mathcal{D}, \mathcal{S})$ is an initial model of an F-logic program \mathcal{P} iff there is a skolemization, $\Pi_{\mathcal{D}}(\mathcal{P})$, of \mathcal{P} such that $\mathcal{S} = \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$.*

Proof. By Theorem 1, if $\mathcal{S} = \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$ then $\mathcal{I} = (\mathcal{D}, \mathcal{S})$ is an initial model of \mathcal{P} . Conversely, if $\mathcal{I} = (\mathcal{D}, \mathcal{S})$ is an initial model of \mathcal{P} , then there is a skolemization, $\Pi_{\mathcal{D}}(\mathcal{P})$, of \mathcal{P} such that $\mathcal{S} \models \Pi_{\mathcal{D}}(\mathcal{P})$, by Definition 9. Since $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$ is the minimal model of $\Pi_{\mathcal{D}}(\mathcal{P})$, it follows that $\mathcal{S} \supseteq \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$. Therefore, we must have $\mathcal{S} = \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(\mathcal{P})})$, for otherwise we get a contradiction with the assumption that \mathcal{I} is an initial model.

Corollary 2. *All initial models of an F-logic program are isomorphic.*

Proof. Let $\mathcal{I} = (\mathcal{D}_1, \mathcal{S}_1)$ and $\mathcal{J} = (\mathcal{D}_2, \mathcal{S}_2)$ be two initial models of an F-logic program P. It follows that there are two skolemizations, $\Pi_{\mathcal{D}_1}(P)$ and $\Pi_{\mathcal{D}_2}(P)$, such that $\mathcal{S}_1 = \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}_1}(P)})$ and $\mathcal{S}_2 = \text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}_2}(P)})$, by Corollary 1. We can construct an 1-1 mapping $\tau : \mathcal{D}_1 \rightarrow \mathcal{D}_2$ such that $\tau(\Pi_{\mathcal{D}_1}(P))$ is the same as $\Pi_{\mathcal{D}_2}(P)$. It can be shown that $\tau(\mathcal{S}_1) = \mathcal{S}_2$ by a transfinite induction on the program consequence operator. Therefore \mathcal{I} and \mathcal{J} are isomorphic.

5 Two Flavors of Entailment

To answer queries and to reason about logical statements we need to define the notion of *logical entailment* of one scoped formula by another. We show that two different notions of entailment are possible for RDF: *strict entailment*, denoted \models , and *relaxed entailment*, denoted \models_r . In general these two entailments have rather different properties, but they coincide for certain classes of programs. Only one of these notions, *i.e.*, relaxed entailment, is currently reflected in the proposal for the RDF model theory [15].

The first type of entailment is defined similarly to classical logic.

Definition 12 (Strict Entailment). *Let ϕ and ψ be two scoped formulas. We write $\phi \models \psi$ iff for every interpretation \mathcal{I} , whenever $\mathcal{I} \models \phi$ then it is the case that $\mathcal{I} \models \psi$.*

This definition has the normally expected properties, *e.g.*, if $\phi \models \psi$ then $\phi \models \psi \vee \xi$, and if $\phi \models \psi \wedge \xi$ then $\phi \models \psi$ and $\phi \models \xi$. As a special case, if ϕ represents an RDF graph, *i.e.*, ϕ is just a conjunction of scoped atomic F-logic formulas, and ψ represents a subgraph of ϕ , then $\phi \models \psi$. This property is analogous to the one exhibited by the current proposal for the RDF model theory [15]. However, the difference is that strict entailment does not hold between a *proper instance*⁷ of an RDF graph and the graph itself. For instance, $\{ \text{john}[\text{likes} \rightarrow \text{food}] \}$ is a proper instance of both $\{ _ \# [\text{likes} \rightarrow \text{food}] \} \wedge \{ _ \# [\text{likes} \rightarrow \text{food}] \}$ and $\{ \text{john}[\text{likes} \rightarrow _ \#] \} \wedge \{ \text{john}[\text{likes} \rightarrow _ \#] \}$, but

$$\begin{aligned} \{ \text{john}[\text{likes} \rightarrow \text{food}] \} &\not\models \{ _ \# [\text{likes} \rightarrow \text{food}] \} \wedge \{ _ \# [\text{likes} \rightarrow \text{food}] \} \\ \{ \text{john}[\text{likes} \rightarrow \text{food}] \} &\not\models \{ \text{john}[\text{likes} \rightarrow _ \#] \} \wedge \{ \text{john}[\text{likes} \rightarrow _ \#] \} \end{aligned}$$

Indeed, when applied to RDF graphs, strict entailment corresponds to isomorphic embedding of graphs (the entailed graph is the one that is embedded). In isomorphic embedding, named resource nodes in one graph are mapped to identically named nodes in another graph, while blank nodes are mapped to blanked nodes. Moreover, this mapping is 1-1.

⁷ A proper instance of an RDF graph is obtained by replacing one or more anonymous resources with named resources [15] (which in our case means replacing them with the constants in \mathcal{C}).

In contrast, the notion of entailment defined in the proposed RDF model theory [15] corresponds to a more relaxed notion of embedding — one where blank nodes can be mapped to anything. In this notion, two blank nodes can even be spliced into the same node.

It turns out that this second notion of entailment, *i.e.*, relaxed entailment, can also be captured in our framework. First, we need the following transformation, which replaces anonymous ID symbols in a scoped formula with existential quantifiers. Let ψ be a scoped formula:

1. If ψ has an occurrence of $_#_$, then replace ψ with $\exists X(\psi')$, where X is a new variable that does not occur anywhere else and ψ' is obtained from ψ by replacing this particular occurrence of $_#_$ with X .
2. If ψ has an occurrence of a numbered anonymous ID symbol, say $_#_1$, then rewrite ψ into $\exists X(\psi')$, where X is a new variable and ψ' denotes ψ where all occurrences of $_#_1$ within the same scope are replaced with X .

Let $\text{strip}(\phi)$ denote the first order formula obtained by an exhaustive application of the above rewrite rules to a given scoped formula ϕ . Note that the new existential quantifiers generated by the rewrite rules are placed at the beginning of the newly created formula. For instance, let $\phi = \forall Y(\text{o}[m(Y) \rightarrow _#_] \leftarrow \text{o}[f(Y) \rightarrow v])$, then $\text{strip}(\phi) = \exists X \forall Y(\text{o}[m(Y) \rightarrow X] \leftarrow \text{o}[f(Y) \rightarrow v])$.

Definition 13 (Relaxed Entailment). *Let P and Q be two F-logic programs. We write $P \approx Q$ iff $P \models \text{strip}(Q)$.*

We will now investigate the relationship between strict and relaxed entailment.

Theorem 2. *Let P and Q be two F-logic programs. If $P \models Q$ then $P \approx Q$.*

Proof. Suppose $P \models Q$. Let $\mathcal{I} \models P$. Then it must be the case that $\mathcal{I} \models Q$. This means that there is a skolemization of Q such that $\mathcal{I} \models \Pi_{\mathcal{D}}(Q)$, where the last \models denotes entailment in the classical sense (note that $\Pi_{\mathcal{D}}(Q)$ has no anonymous ID symbols). Let ν_Q denote the mapping defined by the skolemization: it assigns a constant to each occurrence of an anonymous ID symbol in accordance with the scoping rules. Let η_Q denote the 1-1 and onto mapping from the occurrences of the anonymous ID symbols in Q to the occurrences of the newly generated variables in $\text{strip}(Q)$, which is defined by the strip transformation above. Then it is easy to see that $\nu_Q \circ \eta_Q^{-1}$ is a variable assignment for the existential variables in $\text{strip}(Q)$, which converts $\text{strip}(Q)$ into $\Pi_{\mathcal{D}}(Q)$ (after removal of the now defunct existential quantifiers). Therefore, \mathcal{I} is a model of $\text{strip}(Q)$.

For another connection between these two entailments, it is easy to see that if the consequent is an F-logic program that does not use anonymous ID symbols then the two notions of entailment coincide.

Theorem 3. *Let P and Q be two F-logic programs and suppose Q does not contain anonymous ID symbols. Then $P \models Q$ iff $P \approx Q$.*

Proof. The result follows immediately, since in this case $\text{strip}(Q) = Q$.

Recall that under strict entailment, an instance of an RDF graph does not necessarily entail the graph. This property holds, however, for relaxed entailment, which makes it behave similarly to the RDF model theory [15].

Theorem 4. *Given two F-logic programs P and Q, if Q represents an RDF graph, i.e., a conjunction of scoped atomic F-logic formulas, and P is a proper instance of Q, then $P \approx Q$.*

Proof. Recall that a proper instance of an RDF graph is obtained by replacing one or more anonymous resources with arbitrarily named resources [15], which are constants of \mathcal{C} in our logic.

Since $P \models P$, it follows from Theorem 2 that $P \approx P$, i.e., $P \models \text{strip}(P)$. But $\text{strip}(P)$ is nothing but $\text{strip}(Q)$ with some constants substituted for existential quantifiers. By classical results about entailment, we conclude that $\text{strip}(P) \models \text{strip}(Q)$. Combined with $P \models \text{strip}(P)$, we get $P \models \text{strip}(Q)$, i.e., $P \approx Q$.

Let us revisit the previous example where $\{ \text{john}[\text{likes} \rightarrow \text{food}] \}$ is a proper instance of both $\{ _ \# [\text{likes} \rightarrow \text{food}] \} \wedge \{ _ \# [\text{likes} \rightarrow \text{food}] \}$ and $\{ \text{john}[\text{likes} \rightarrow _ \#] \} \wedge \{ \text{john}[\text{likes} \rightarrow _ \#] \}$. We have

$$\begin{aligned} \{ \text{john}[\text{likes} \rightarrow \text{food}] \} &\approx \{ _ \# [\text{likes} \rightarrow \text{food}] \} \wedge \{ _ \# [\text{likes} \rightarrow \text{food}] \} \\ \{ \text{john}[\text{likes} \rightarrow \text{food}] \} &\approx \{ \text{john}[\text{likes} \rightarrow _ \#] \} \wedge \{ \text{john}[\text{likes} \rightarrow _ \#] \} \end{aligned}$$

We should note that it is not clear which notion of entailment, \models or \approx , is more appropriate for RDF graph entailment. Perhaps, both should be used, as they give rise to different notions of equivalence for these graphs. We say that $P \equiv Q$ iff $P \models Q$ and $Q \models P$. Similarly, $P \cong Q$ iff $P \approx Q$ and $Q \approx P$. It is easy to show that if $P \equiv Q$ then the initial models of P and Q are isomorphic and in a well-defined sense they correspond to the RDF graph represented by these programs. In contrast, if $P \cong Q$, then P and Q can still have non-isomorphic initial models and their RDF graphs can be different. For instance, if P has only one fact, $\{ a[b \rightarrow c] \}$, and Q is $\{ a[b \rightarrow c], a[b \rightarrow _ \#] \}$, then $P \cong Q$. However, the RDF graph of P has only one arc and two nodes, while the graph of Q has three nodes and two arcs. But each graph can be (non-isomorphically) embedded into the other. The relationship between relaxed entailment and non-isomorphic graph embedding is explored in the following theorem.

Theorem 5. *Let P and Q be F-logic programs representing RDF graphs. Let $\mathcal{I} = (\mathcal{D}_1, \mathcal{R})$ and $\mathcal{J} = (\mathcal{D}_2, \mathcal{S})$ be the initial models of P and Q, respectively. Then $P \approx Q$ iff there is an augmented domain mapping $\lambda: \mathcal{D}_2 \rightarrow \mathcal{D}_1 \cup \mathcal{C}$ such that $\lambda(\mathcal{S}) \subseteq \mathcal{R}$.*

Proof. Suppose $P \approx Q$. Then $\mathcal{I} \models \text{strip}(Q)$ by Definition 13. This means that there is an assignment, ρ , of the existential variables in $\text{strip}(Q)$ to the elements in $\mathcal{C} \cup \mathcal{D}_1$ such that $\mathcal{R} \models Q'$, where Q' is $\rho(\text{strip}(Q))$ ($\text{strip}(Q)$ with the now defunct existential quantifiers removed).

Let η_Q denote the 1-1 and onto mapping from the occurrences of the anonymous ID symbols in Q to the occurrences of the newly generated variables in $\text{strip}(Q)$, which is defined by the `strip` transformation. Then $\rho(\eta_Q(Q)) = Q'$.

Since $\mathcal{J} = (\mathcal{D}_2, \mathcal{S})$ is an initial model of Q , let ν_Q denote the mapping defined by the skolemization: it assigns a constant to each occurrence of an anonymous ID symbol in accordance with the scoping rules. Because Q represents an RDF graph, *i.e.*, a set of facts, it follows that $\rho(\eta_Q(\nu_Q^{-1}(\mathcal{S}))) = Q'$. Finally, since $\mathcal{R} \models Q'$ and both \mathcal{R} and Q' are sets of facts, it follows that $Q' \subseteq \mathcal{R}$. Therefore, $\rho \circ \eta_Q \circ \nu_Q^{-1}$ is the required augmented domain mapping.

By a similar argument we can also prove the opposite direction.

To conclude our discussion of entailment, we would like to remark on the relationship between equality and the semantics presented in Sections 4 and 5. It is a rather common misconception that a Herbrand style semantics, such as the one presented here, implies a closed world assumption or a unique name assumption. In actuality, resolution-based proof theory for full classical predicate calculus relies on Herbrand models. This semantics can be extended to include any kind of equality theory by superimposing an equivalence relation over the Herbrand universe [8]. Likewise, our semantics for anonymous identity implies neither the unique name nor the closed world assumption. Equality theories can be incorporated in the same way as in [8] and the semantics can be made nonmonotonic if required. These extensions are implemented in the `FLORA-2` system [33].

6 Compositionality of the Semantics

It is our contention that the semantics for Semantic Web languages should have the *semantic compositionality* property. Informally, this means that the result of putting together, say, two RDF documents should be another valid RDF document. In other words, if P_1 and P_2 are two RDF documents then each model of $P_1 \cup P_2$ should be some sort of a union of the models of P_1 and P_2 separately.

It turns out that the F-logic language with anonymous ID symbols presented in the previous section satisfies this requirement, whereas the N3 notation for RDF, similar to RDF triple syntax (see [3] for an introduction to N3), and the current proposal for the RDF model theory [15] do not. To make this requirement precise, we first need to define what we mean by the “union” of two interpretations.

First, we recall the notion of a disjoint union of sets. Let \mathcal{D}_1 and \mathcal{D}_2 be two sets. Their *disjoint union*, denoted $\mathcal{D}_1 \uplus \mathcal{D}_2$, is obtained by “renaming apart” the common elements of \mathcal{D}_1 and \mathcal{D}_2 (thus making the sets disjoint) and then taking the union. The set $\mathcal{D}_1 \uplus \mathcal{D}_2$ has the following properties:

- There are 1-1 mappings $\iota_1 : \mathcal{D}_1 \rightarrow \mathcal{D}_1 \uplus \mathcal{D}_2$ and $\iota_2 : \mathcal{D}_2 \rightarrow \mathcal{D}_1 \uplus \mathcal{D}_2$;
- $\mathcal{D}_1 \uplus \mathcal{D}_2 = \iota_1(\mathcal{D}_1) \cup \iota_2(\mathcal{D}_2)$; and
- $\iota_1(\mathcal{D}_1) \cap \iota_2(\mathcal{D}_2) = \emptyset$.

Definition 14 (Disjoint Union of Interpretations). Let $\mathcal{I}_1 = (\mathcal{D}_1, \mathcal{S}_1)$ and $\mathcal{I}_2 = (\mathcal{D}_2, \mathcal{S}_2)$ be two interpretations. A disjoint union of \mathcal{I}_1 and \mathcal{I}_2 , denoted $\mathcal{I}_1 \uplus \mathcal{I}_2$, is an interpretation of the form $(\mathcal{D}_1 \uplus \mathcal{D}_2, \iota_1(\mathcal{S}_1) \cup \iota_2(\mathcal{S}_2))$, where ι_1 and ι_2 are the 1-1 embeddings $\iota_1 : \mathcal{D}_1 \rightarrow \mathcal{D}_1 \uplus \mathcal{D}_2$ and $\iota_2 : \mathcal{D}_2 \rightarrow \mathcal{D}_1 \uplus \mathcal{D}_2$ that are associated with $\mathcal{D}_1 \uplus \mathcal{D}_2$. In other words, a disjoint union of two interpretations makes sure that anonymous constants that happen to occur in both anonymous domains \mathcal{D}_1 and \mathcal{D}_2 are renamed apart (both in these domains and in \mathcal{S}_1 and \mathcal{S}_2) prior to taking the union.

We can now state the following simple result:

Theorem 6. Let P_1 and P_2 be two F-logic programs, and let \mathcal{I}_1 and \mathcal{I}_2 be the initial models of P_1 and P_2 , respectively. Suppose that both P_1 and P_2 represent an RDF graph (i.e., a conjunction of scoped atomic formulas). Then $\mathcal{I}_1 \uplus \mathcal{I}_2$ is an initial model of $P_1 \cup P_2$.

Proof. Since P_1 and P_2 represent RDF graphs, they may contain anonymous ID symbols but no variables. Let $\mathcal{I}_1 = (\mathcal{D}_1, \mathcal{S}_1)$ and $\mathcal{I}_2 = (\mathcal{D}_2, \mathcal{S}_2)$. To simplify the exposition, let us assume that $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$.

There must be skolemizations $\Pi_{\mathcal{D}_1}(P_1)$ and $\Pi_{\mathcal{D}_2}(P_2)$, such that \mathcal{S}_1 and \mathcal{S}_2 are, respectively, the minimal models of $\Pi_{\mathcal{D}_1}(P_1)$ and $\Pi_{\mathcal{D}_2}(P_2)$. Since $\mathcal{D}_1 \cap \mathcal{D}_2 = \emptyset$, it follows that $\Pi_{\mathcal{D}_1}(P_1) \cup \Pi_{\mathcal{D}_2}(P_2)$ is a valid skolemization of $P_1 \cup P_2$. Since P_1 and P_2 represent RDF graphs, the skolemizations $\Pi_{\mathcal{D}_1}(P_1)$ and $\Pi_{\mathcal{D}_2}(P_2)$ have no variables and thus \mathcal{S}_1 and \mathcal{S}_2 are identical to $\Pi_{\mathcal{D}_1}(P_1)$ and $\Pi_{\mathcal{D}_2}(P_2)$, respectively. It is therefore easy to see that $\mathcal{S}_1 \cup \mathcal{S}_2$ is the minimal model of $\Pi_{\mathcal{D}_1}(P_1) \cup \Pi_{\mathcal{D}_2}(P_2)$. Thus, $\mathcal{I}_1 \uplus \mathcal{I}_2$ is an initial model of $P_1 \cup P_2$.

The above result could be extended to programs that contain inference rules. The main difference is that $P_1 \cup P_2$ can derive more facts than the union of the facts that P_1 and P_2 can derive in separation. Consequently, $\mathcal{I}_1 \uplus \mathcal{I}_2$ may not be an initial model of $P_1 \cup P_2$, but rather a congruent sub-interpretation of an initial model.

It appears that the current proposal for the RDF model theory does not have the compositionality property. To see this, consider the sentence *someone loves Mary* represented using RDF triple syntax:

$$\{[\text{loves}], [X], [\text{Mary}]\}$$

Let P_1 denote this document. Let P_2 denote the document that represents the sentence *someone invented the bulb*:

$$\{[\text{inventor}], [X], [\text{Bulb}]\}$$

By composing the two documents, we get $P_1 \cup P_2$:

$$\begin{aligned} &\{[\text{loves}], [X], [\text{Mary}]\} \\ &\{[\text{inventor}], [X], [\text{Bulb}]\} \end{aligned}$$

Although the interpretation $\{ \{[\text{loves}], [\text{John}], [\text{Mary}]\} \}$ is a model for the document P_1 and $\{ \{[\text{inventor}], [\text{ThomasEdison}], [\text{Bulb}]\} \}$ is a model for P_2 , their (disjoint) union, $\{ \{[\text{loves}], [\text{John}], [\text{Mary}]\}, \{[\text{inventor}], [\text{ThomasEdison}], [\text{Bulb}]\} \}$ is *not* a model of $P_1 \cup P_2$.

This problem can be rectified by adding the notion of scope, as introduced in this paper, to N3 and the RDF model theory. An alternative solution — changing the definition of the union of RDF graphs — would be undesirable, as it will only serve to highlight the inadequacy of the current syntax and semantics of N3.

7 Properties of Reification

In this section we discuss some properties of reification in F-logic and compare them to the current proposal for RDF model theory [15].

One important difference is that in F-logic reified statements are themselves objects, whereas in the RDF model theory they are referred to by names. In particular, one can give two names to the same statement and these would be completely unrelated objects. In our opinion, such a semantics is too weak.⁸ To illustrate the problem, consider again the statement from Section 3 that *John believes that Thomas Edison (this time a known resource) invented the bulb*:

```
{[type], [X], [RDF:Statement]}
{[predicate], [X], [inventor]}
{[subject], [X], [http://foo.org/TheBulb]}
{[object], [X], [http://foo.org/ThomasEdison]}

{believes, [http://xyz.com/John], [X]}
```

In the RDF triple syntax, one can add another reference to the reified statement:

```
{[type], [Y], [RDF:Statement]}
{[predicate], [Y], [inventor]}
{[subject], [Y], [http://foo.org/TheBulb]}
{[object], [Y], [http://foo.org/ThomasEdison]}
```

However, in RDF, the objects referred to by X and Y would have nothing to do with each other. For instance, John would believe X but not Y. Likewise, stating that X is a true statement or that it was made by Encyclopedia Britannica does not imply that the same holds for Y. In contrast, in F-logic we can say

```
'http://foo.org/TheBulb'[inventor → 'http://foo.org/ThomasEdison']
[
  veracity → true,
  authority → 'http://www.britannica.com/'
].
```

⁸ There has been lengthy discussion of this problem on the RDF mailing list <http://lists.w3.org/Archives/Public/www-rdf-interest/> and it has been proposed that reifications of RDF statements should be viewed as *statings* — which are “instances” of statements. (See <http://ilrt.org/discovery/2000/11/statements/>.) However, we remain unconvinced as to the desirability of this distinction.

So anyone who believes this statement would be believing a statement whose veracity attribute has the value `true` and which is believed to be authorized by `Britannica`. In fact, we can even go further and specify the rule

$$\text{Statement} \leftarrow \text{Statement}[\text{veracity} \rightarrow \text{true}].$$

which will make any statement whose veracity property is “true” into a true statement in every model. Thus, for example, Mary, who also believes this statement and who might even be defined in a different document, would be believing a true assertion.

Note that in our semantics conjunctions of atomic formulas are terms and thus can be treated as objects. In fact, this idea can be further extended to allow reification of more general statements. We will show some simple uses of reified conjunctions. First, for some properties, such as beliefs, it is reasonable to assume that conjunctions can be broken apart, because if someone believes in a combined statement then she is likely to believe in all of its parts. This can be expressed by the following rule:

$$X[\text{believes} \rightarrow S1], X[\text{believes} \rightarrow S2] \leftarrow X[\text{believes} \rightarrow (S1 \wedge S2)].$$

On the other hand, in some contexts reified conjuncted statements cannot be broken up. For instance, the following might be considered as a true statement

$$(\text{john}[\text{likes} \rightarrow \text{sally}] \wedge \text{sally}[\text{likes} \rightarrow \text{john}]) [\text{statementAbout} \rightarrow \text{friendship}].$$

But

$$(\text{john}[\text{likes} \rightarrow \text{sally}]) [\text{statementAbout} \rightarrow \text{friendship}]$$

is not necessarily a true statement.

An important advantage of our semantics is that it is developed in a general framework, which provides a basis for reasoning about reified statements — not only for stating them as facts. We have already shown how one can state that certain beliefs are actually true. However, there are many more possibilities for non-trivial reasoning. For instance, *Bob always believes when Alice says that some statement is made by a trusted third party*. This knowledge can be encoded using the following rule:

$$\begin{aligned} \text{'http://xyz.com/Bob'}[\text{believes} \rightarrow \text{Statement}[\text{authority} \rightarrow A]] \leftarrow \\ \text{'http://xyz.com/Alice'}[\text{says} \rightarrow \text{Statement}[\text{authority} \rightarrow A]], \\ \text{'http://xyz.com/Bob'}[\text{trusts} \rightarrow A]. \end{aligned}$$

In fact, one can express a number of belief systems using rules and then combine them with reification to derive useful conclusions about reified beliefs.

8 Reification and Logical Paradoxes

Year 2001 marked a Centennial of the famous Russell’s paradox — a remarkable discovery of the inconsistency of Frege’s comprehension axioms. The basic reason

for the existence of this paradox in Frege’s logic is the ability to reify logical sentences and make statements about these sentences. Ever since this discovery logics that permit reification were subject to strict and just scrutiny. In this section we will briefly explain the nature of Russell’s paradox as it pertains to reification and how it affects our theory. A nice discussion of this paradox and of the ways to avoid it appears in [27].

The simplest and most familiar form of Russell’s paradox is the so called Liar’s paradox — a statement, L , which says that L is false. If L is true then it must be false, and if it is false then it must be true. In order to be able to make such statements within a logic we need the ability to refer to logical sentences in the language of the logic. We will show how such a paradox can be constructed within F-logic extended with the reification mechanism. In fact, we shall see that paradoxes can be constructed using HiLog alone.

To obtain a paradox, we need the so called *comprehension axiom schema* of the following form

$$\exists P \forall X (P(X) \leftrightarrow \phi(X))$$

in addition to reification. Here ϕ is some formula that does not have P as a free variable. This axiom says that we can always define a predicate that is true precisely of those things that have the property ϕ . For instance, $\phi(X)$ can be something like $q(X) \wedge X$.

Note that the comprehension axioms are valid HiLog formulas. If we now substitute $\neg X(X)$ for ϕ and choose X to be P , we get

$$\exists P (P(P) \leftrightarrow \neg P(P))$$

which is unsatisfiable. The paradox consists in the realization that a very natural set of axioms, such as comprehension axioms, is unsatisfiable.

We might still think that Russell’s paradox is no more than a curiosity, because the comprehension axioms are so general that they are, perhaps, of little use for knowledge representation on the Semantic Web. Unfortunately we are not out of the woods yet. Let *true* be a new predicate symbol and consider the following *truth axiom*:

$$\forall X (\text{true}(X) \leftrightarrow X) \tag{1}$$

This axiom states that everything that is asserted to be true in the meta-level (*i.e.*, using the truth-predicate applied to reified formulas) is, indeed, true in the logic. It turns out that this axiom is an instance of the comprehension axiom schema and it is almost as bad. Unlike the comprehension axioms, we cannot simply dismiss the truth axiom as too general because it is so close to routine things that people do in knowledge representation. For instance, one might want to say that Alice does not know false facts ($X : \neg \text{sallyKnows}(X)$) and that Bob knows every true fact believed by Sally ($\text{bobKnows}(X) : \neg X, \text{sallyBelieves}(X)$).⁹

To see how the truth axiom leads to a paradox, we assume that we are allowed to define new concepts using previously defined ones. Specifically, let r

⁹ Another example is the axiom $\text{Statement} \leftarrow \text{Statement}[\text{veracity} \rightarrow \text{true}]$ of the previous section.

be a completely new symbol defined as follows:

$$\forall X (r(X) \leftrightarrow \neg \text{true}(X(X)))$$

Such definitions are the staple of every logic-based knowledge representation language, although in some (like Prolog) the definition would be unidirectional.

By substituting r for X , we obtain the so called *heterological Liar's statement*:

$$r(r) \leftrightarrow \neg \text{true}(r(r)) \tag{2}$$

It is easy to see that the Liar's statement is inconsistent with the truth axiom, since we can derive a contradiction:

$$r(r) \leftrightarrow \neg \text{true}(r(r)) \leftrightarrow \neg r(r)$$

Since the truth axiom is simply a pair of Horn clauses and the Liar's statement is also not too far off from what an arbitrary logic program could have, we need to re-examine our reified F-logic to see if it is of any use at all. Is it possible that an innocuous rule set written by an unsuspecting provider of Semantic Web content can have a hidden inconsistency?

Theorem 7. *Horn programs in reified F-logic augmented with the truth axiom (1) are consistent.*

Proof. Let P be an F-logic Horn program. Since the truth axiom is just a pair of Horn clauses, we can consider them to be part of P . By Theorem 1, P has an initial model and therefore is consistent.

The above theorem can be extended to programs that have negative literals in the rule body under the well-founded semantics [18]. Therefore, we can be satisfied that for most practical purposes reified F-logic is an adequate knowledge representation formalism.

An interesting question is whether it is possible to extend the scope of reification in F-logic to include disjunction and negation. The answer to the former is yes, because disjunctive logic programming [26] can be adapted to F-logic. On the other hand, reifying negation would create problems unless the structure of the programs and the truth axiom are restricted in some way. Perlis [27] proposed an elegant way to restrict the truth axiom, which eliminates paradoxes from first-order logics.¹⁰ To see how inconsistency can arise if negation is reified, consider the following program:

$$\begin{aligned} \text{true}(\neg p(X)) &\leftarrow p(X). \\ p(X) &\leftarrow \neg p(X). \end{aligned}$$

¹⁰ Unfortunately, this solution does not quite apply to HiLog, because Perlis' syntax precludes the heterological Liar's statement.

Because of the truth axiom, we obtain $p(X) \leftrightarrow \neg p(X)$.¹¹

To summarize our discussion, we have shown that reified F-logic avoids paradoxes through the following restrictions:

- Programs must be sets of extended Horn clauses (*i.e.*, no negation in the rule head); and
- Reification of negative facts is not permitted, which stops negation from appearing in the rule heads through the back door.

This is not the only useful set of restrictions, however. In the $\mathcal{F}LORA-2$ system [33], reification of negative facts is allowed and thus the second condition above does not hold. Instead, $\mathcal{F}LORA-2$ precludes negative literals in rule heads (like the first restriction above) and closes the remaining loophole with the following restriction:

The head of a rule cannot be a variable term.

Note that rules like $X(Y) :- \text{body}$ or $X[Y \rightarrow Z] :- \text{body}$ are still allowed. The excluded rules are of the form $X :- \text{body}$. This restriction effectively rules out the truth axiom and, since negative literals cannot occur in the rule head, it becomes impossible to derive such facts despite the fact that individual variables can be bound to negative reified facts.

9 Conclusion

In this paper we presented an extension of F-logic that supports anonymous resources and reification. Such an extension makes F-logic a suitable foundation for Semantic Web languages, such as RDF. The language has a simple and natural, paradox-free semantics and a proof theory, and has been implemented in the $\mathcal{F}LORA-2$ system [33]. In particular, our semantics rectifies a number of drawbacks of the current proposal for the RDF model theory [15]: non-compositional semantics and weaker than necessary treatment of reification. We also pointed out that there are at least two different (and useful) meanings for the notion of RDF graph entailment. On top of this, our semantics is much more general than [15], as it is given in the framework of an expressive rule-based and frame-based language, which opens up many possibilities for encoding knowledge.

Anonymous objects and reification have uses outside of the Semantic Web context. In fact, they were originally introduced in $\mathcal{F}LORA-2$ to satisfy database-specific needs. For instance, anonymous objects are related to the so-called *pure values* in object-oriented databases [1, 23] and they are also very convenient for object-oriented modeling of XML documents. In addition, they turned out to be valuable from the software engineering point of view. Reification was introduced

¹¹ It should be kept in mind that this contradiction arises only in classical logic, while logic programs (especially those that involve negation) are based on non-classical logic. However, this example shows that developing an appropriate semantics for reified negation has issues to overcome.

in *FLORA-2* to provide a clean semantics (and an implementation) for aggregate operators, which take a query (*i.e.*, a reified formula) as an argument and perform summarization operations over the set of answers to the query.

Acknowledgement

The authors would like to thank the anonymous referees for their comments and suggestions that help improve this paper.

This work was supported in part by NSF grant IIS-0072927. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing those of the funding agency or the U.S. government.

References

1. F. Bancilhon, C. Delobel, and P. Kanellakis, editors. *Building an Object-Oriented Database System: The Story of O2*. Morgan Kaufmann, San Francisco, CA, 1990.
2. T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifiers (URI): Generic syntax, August 1998. <http://www.isi.edu/in-notes/rfc2396.txt>.
3. Tim Berners-Lee. Primer: Getting into RDF & Semantic Web using N3. <http://www.w3.org/2000/10/swap/Primer>.
4. A.J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133:205–265, October 1994.
5. A.J. Bonner and M. Kifer. A logic for programming database transactions. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, March 1998.
6. T. Bray, D. Hollander, and A. Layman. Namespaces in XML, January 1999. <http://www.w3.org/TR/REC-xml-names/>.
7. J. Broekstra, C. Fluit, and F. van Harmelen. The state of the art on representation and query languages for semistructured data. Technical report, Administrator, Nederland BV, August 2000.
8. C. L. Chang and R. C. T. Lee. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, 1973.
9. W. Chen, M. Kifer, and D.S. Warren. HiLog: A first-order semantics for higher-order logic programming constructs. In *North American Conference on Logic Programming*, Cambridge, MA, October 1989. MIT Press.
10. W. Chen, M. Kifer, and D.S. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.
11. W. Conen, R. Klapsing, and E. Köppen. RDF M&S revisited: From reification to nesting, from containers to lists, from dialect to pure XML. In *Semantic Web Working Symposium (SWWS)*, 2001.
12. H. Davulcu, G. Yang, M. Kifer, and I.V. Ramakrishnan. Design and implementation of the physical layer in webbases: The XRouter experience. In *First International Conference on Computational Logic, DOOD'2000 Stream*, July 2000.
13. S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for RDF. In *QL'98 - The Query Languages Workshop*, December 1998.
14. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In R. Meersman et al., editor, *Database Semantics, Semantic Issues in Multimedia Systems*, pages 351–369. Kluwer Academic Publisher, Boston, 1999.

15. P. Hayes (editor). RDF Model Theory. Technical report, W3C, April 2002. <http://www.w3.org/TR/rdf-mt/>.
16. D. Fensel, S. Decker, M. Erdmann, and R. Studer. Ontobroker: Or how to enable intelligent access to the WWW. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1998.
17. R. Fikes and D.L. McGuinness. An axiomatic semantics for RDF, RDF Schema, and DAML+OIL. Technical Report KSL-01-01, Knowledge Systems Laboratory, Stanford University, October 2001.
18. A. Van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, 38(3):620–650, July 1991.
19. M.R. Genesereth. Knowledge interchange format. Technical Report NCITS.T2/98-004, Knowledge Systems Laboratory, Stanford University, 1998. Draft proposed American National Standard, <http://logic.stanford.edu/kif/dpans.html>.
20. C. H. Goh, S. Bressan, S. E. Madnick, and M. D. Siegel. Context interchange: Representing and reasoning about data semantics in heterogeneous systems. Technical report, MIT, School of Management, 1996.
21. A. Gupta, B. Ludäscher, and M. E. Martone. Knowledge-based integration of neuroscience data sources. In *12th International Conference on Scientific and Statistical Database Management (SSDBM)*, Berlin, Germany, July 2000. IEEE.
22. G.-J. Houben. HERA: Automatically generating hypermedia front-ends for ad hoc data from heterogeneous and legacy information systems. In *Engineering Federated Information Systems*, pages 81–88. Aka and IOS Press, 2000.
23. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.
24. O. Lasilla and R. R. Swick (editors). Resource description framework (RDF) model and syntax specification. Technical report, W3C, February 1999. <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/>.
25. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1984.
26. J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, Massachusetts, 1992.
27. D. Perlis. Languages with self-reference i: Foundations. *Artificial Intelligence*, 25:301–322, 1985.
28. M. Sintek and S. Decker. TRIPLE – An RDF query, inference, and transformation language. In *Deductive Databases and Knowledge Management (DDL’2001)*, October 2001.
29. S. Staab, J. Angele, S. Decker, M. Erdmann, A. Hotho, A. Maedche, H.-P. Schnurr, R. Studer, and Y. Sure. AI for the Web — Ontology-based community web portals. In *9-th International World Wide Web Conference (WWW9)*, Amsterdam, The Netherlands, May 2000.
30. G. Yang and M. Kifer. Implementing an efficient DOOD system using a tabling logic engine. In *First International Conference on Computational Logic, DOOD’2000 Stream*, July 2000.
31. G. Yang and M. Kifer. Well-founded optimism: Inheritance in frame-based knowledge bases. In *International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE)*, 2002.
32. G. Yang and M. Kifer. Inheritance and rules in object-oriented semantic web languages. In *International Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML)*, 2003.
33. G. Yang, M. Kifer, and C. Zhao. FLORA-2: User’s manual. <http://flora.sourceforge.net/>, June 2003.