

On the Semantics of Anonymous Identity and Reification^{*}

Guizhen Yang Michael Kifer

Department of Computer Science
Stony Brook University
Stony Brook, NY 11794, U.S.A.
{guizyang,kifer}@CS.StonyBrook.EDU

Abstract. Reification and anonymous resources 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 [18], 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 [12, 10].

In this paper we propose a simple semantics for reification and anonymous resources in F-logic [17] — a frame-based logic language, which is a popular formalism for representing and reasoning about semantic information on the Web [22, 9, 11, 8, 7].

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, and deduction, and it has been shown to provide an effective inference service for RDF [8, 21].

1 Introduction

RDF [18] 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 [12, 10].

While much of the RDF syntax is first-order in nature, *reification* (which is involved in expressing statements like “Tom believes that Alice said that RDF is a good idea”) is not.¹ 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 [17] as the underlying formalism.

The choice of F-logic is motivated by several considerations. First, it has been a popular vehicle for information mediation and representing semantic information on the Web whenever complex inference is required [14, 16, 22, 9, 11,

^{*} This work was supported in part by NSF grant IIS-0072927.

¹ In fact, when left to its own devices, reification can lead to logical paradoxes [20].

7]. As such applications grow in complexity, the need for an inferencing service will increase. For instance, [8] envisions F-logic precisely as such a service for RDF.

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 [25]. In fact, we argue that the current proposal for the RDF model theory [10] has several weaknesses, such as *non-compositionality*, 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, but only one is currently reflected in the RDF model theory document [10].

The idea of embedding RDF into a larger theory is, of course, not new. Embedding RDF into F-logic was proposed in [8], and in [12] the same was done for KIF [13]. Both proposals are incomplete, however, as they do not address reification and anonymous resources. The embedding into F-logic *would have been* complete if F-logic, as described in [17], had support for these two features. We are rectifying this situation in the present work. Our proposed semantics is conceptually very simple² and is inspired by HiLog [6] — 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 [23, 24] — a powerful frame-based language for knowledge representation and reasoning, which is based on F-logic, HiLog, and Transaction Logic [2, 3]. Therefore, 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 [23, 24] and TRIPLE [21], 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 [15] to processing semistructured and semantic information on the Web [7].

This paper is organized as follows. Section 2 surveys the necessary background from F-logic and HiLog. Section 3 motivates the proposed extension to F-logic from the point of view of modeling RDF. Section 4 formally treats the semantics of the proposed extensions. Sections 5 and 6 discuss the properties of the semantics introduced in Section 4 and point out some problems with the current proposal for the RDF model theory [10]. Section 7 concludes the paper.

² While it is simple, it is not obvious, as a number of authors believed that F-logic does not generalize to deal with reification directly [4, 8, 21].

2 Preliminaries

In this section we review the main ideas behind F-logic [17] and HiLog [6] — 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].
```

Here `spouse → john` is a *single-valued* attribute specification; it says that `mary` has a 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.

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:

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

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 [5, 6] 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`:

$$\text{call}(X) :- X.$$

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 that Mary likes RDF (and possibly other beliefs) as follows:

$$\text{believes}(\text{bob}, \text{likes}(\text{mary}, \text{rdf})).$$

and then state that whatever Bob believes is true:

$$X :- \text{believes}(\text{bob}, X).$$

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:

$$\begin{aligned} \text{closure}(\text{Pred})(X, Y) & :- \text{Pred}(X, Y). \\ \text{closure}(\text{Pred})(X, Y) & :- \text{Pred}(X, Z), \text{closure}(\text{Pred})(Z, Y). \end{aligned}$$

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 [6].

When combined with F-logic, HiLog enables powerful meta-programming features [17, 23, 24]. 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 [17, 23], but its use for supporting reification has not been considered.

3 Supporting RDF in F-logic

It was argued in [8] that F-logic is a natural formalism to provide semantics and inference service for RDF(S) [18]. However, some important aspects, such as anonymous resources, containers, and reification were left out because the original F-logic [17] 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 Section 4.

First, we briefly recall the RDF data model. An RDF document is a finite set of statements of the form {**predicate**, **subject**, **object**}, where **predicate** is a *property*, **subject** is a *resource*, and **object** is a resource or a literal.

A resource describes a real or conceptual entity (*e.g.* “John Doe”). Typically, resources are represented as URIs. 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. A property is a predicate that specifies a binary relationship (*e.g.* “works for”). In RDF, properties form a subset of resources and are also represented by URIs. Finally, literals are constants in some primitive data types, such as a **string** or **number**.

A set of RDF statements can 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 by **p**.

3.1 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 by double quotes denotes a literal. In F-logic, the same statement is written like this:

'<http://foo.org/TheBulb>'[inventor \rightarrow 'Thomas Edison'].

One difficulty arises in translating 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 the 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 [17], but were introduced in *F_LORA-2* [24, 23] — our implementation of F-logic that extends it with many additional concepts. To represent such objects, *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 *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:

```

_#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.

3.2 Reified Statements

Reification in RDF is used to make 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 the RDF triple syntax one would have to write a rather cumbersome document as follows:

```

{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 → -#[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 6 that this syntax and its corresponding semantics is superior to that of the current proposal for RDF model theory [10]. It also permits more interesting reasoning to be easily performed over reified statements (see Section 6).

We should note that the above syntax is *not* the actual syntax of $\mathcal{F}\text{LORA-2}$ [24]. It is also quite different from the F-logic syntax as described in [17]. We do so to avoid introducing additional features of F-logic and $\mathcal{F}\text{LORA-2}$ and to simplify the formal development of the model theory in Section 4. In fact, the actual syntax of $\mathcal{F}\text{LORA-2}$ is much richer, which allows to write the above and the earlier sentences more succinctly:

```

-#[
  name → 'John Doe',
  believes →
    $\{'http://foo.org/TheBulb'[inventor → -#[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 → -#1]` and `-#1[name → 'Thomas Edison']`. The interested reader is referred to [24] for 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 [10].

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 $\mathfrak{o}[\mathfrak{m} \rightarrow \mathfrak{v}]$, which specifies that the object \mathfrak{o} has a multivalued method, \mathfrak{m} , whose return value is a set that contains \mathfrak{v} as a member. The symbols \mathfrak{o} , \mathfrak{m} , and \mathfrak{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, $_ \#1, _ \#2, \dots$ (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. While defining semantics for F-logic programs, we will always *fix* an 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 essentially the same but slightly different. 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 intended meaning 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 [5, 6] 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.
- Any unnumbered or numbered ID symbol, $_ \#, _ \#1, _ \#2, \dots$, is a term.
- 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 or a HiLog atom.
- If $\mathfrak{o}, \mathfrak{m}$, and \mathfrak{v} are terms, then $\mathfrak{o}[\mathfrak{m} \rightarrow \mathfrak{v}]$ is a term, also called an F-logic term or an F-logic atom.
- If A_1, \dots, A_n are terms, then their conjunction, $A_1 \wedge \dots \wedge A_n$, is a term.

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 the above Definition 1

indicate that atomic formulas as well as their conjunctions are terms and so 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 [5, 6].

Definition 2 (Flat Formula). *Any HiLog atom or F-logic atom is an atomic flat formula. 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.

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 $\mathcal{C} \cup \mathcal{D}$. In other words, 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 regard to \mathcal{D} . Intuitively, \mathcal{S} represents “what is true” in \mathcal{I} .³

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,

³ Observe 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.

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 is by set inclusion. But it makes sense only for interpretations over the same domain. Another way involves domain mappings and domain isomorphisms. Formally, we have the following definitions.

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

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

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 and only apply to the top level conjuncts and disjuncts. For instance, $\{ _ \#1[\text{loves} \rightarrow \text{mary}] \} \wedge \{ \exists X(_ \#1[\text{child} \rightarrow X]) \}$ is a valid scoped formula whereas $\{ _ \#1[\text{loves} \rightarrow \text{mary}] \} \wedge \exists X(_ \#1[\text{child} \rightarrow X])$ is not. Although we could define even more general scoping rules, including nested scoping, this is probably not useful in practice and so we shall 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 atoms in the rule head (the A_i 's) are 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.

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 regard to \mathcal{D} , denoted $\Pi_{\mathcal{D}}(\phi)$, is a formula obtained as follows:*

- If $\phi = \psi \vee \varphi$, then $\Pi_{\mathcal{D}}(\phi) = \Pi_{\mathcal{D}}(\psi) \vee \Pi_{\mathcal{D}}(\varphi)$.
- If $\phi = \psi \wedge \varphi$, then $\Pi_{\mathcal{D}}(\phi) = \Pi_{\mathcal{D}}(\psi) \wedge \Pi_{\mathcal{D}}(\varphi)$.
- 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:
 - Every occurrence of the unnumbered anonymous ID symbol, $_ \#$, in ψ is mapped to a distinct constant in \mathcal{D} ; and
 - Every different numbered anonymous ID symbol in ψ is mapped to a distinct constant in \mathcal{D} .

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 $_ \#1$, are mapped to the same distinct element of \mathcal{D} . Also observe that two occurrences of the same numbered ID symbol, say $_ \#1$, under different scopes are mapped to different elements. For instance, consider a scoped formula $\{ \phi \} \wedge \{ \psi \}$. Then occurrences of $_ \#1$ in ϕ are going to be skolemized differently from those in ψ .

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

```

_#[_# → _#1], _#1[name → mary].
_#1[name → 'Ora Lassila'], 'RDF'[creator → _#1].

```

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

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

If we map the first occurrence of $_#\$ to $\mathfrak{o}1$, its second occurrence to $\mathfrak{o}4$, both occurrences of $_#\1$ in the first rule to $\mathfrak{o}2$, and both occurrences of $_#\1$ in the second rule to $\mathfrak{o}3$, then we obtain the following skolemization $\Pi_{\mathcal{D}}(P)$:

$$\mathfrak{o}1[\mathfrak{o}4 \rightarrow \mathfrak{o}2], \mathfrak{o}2[\text{name} \rightarrow \text{mary}]. \\ \mathfrak{o}3[\text{name} \rightarrow \text{'Ora Lassila'}, \text{'RDF'}[\text{creator} \rightarrow \mathfrak{o}3].$$

A different way of mapping anonymous ID symbols to the constants in \mathcal{D} (e.g., where the first occurrence of $_#\$ is mapped to $\mathfrak{o}4$ and the second to $\mathfrak{o}1$) 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 $\mathcal{I} \models \psi$, where $\mathcal{I} \models \psi$ is defined in the classical sense:*

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

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 the equivalence.*

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 it 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, this assertion should not be taken literally.

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.

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 regard to \mathcal{D} . Similarly to [19], we can define a program consequence operator, $\mathcal{T}_{\Pi_{\mathcal{D}}(P)}$, which maps an interpretation, $\mathcal{I} = (\mathcal{D}, \mathcal{S})$, over \mathcal{D} to another interpretation over \mathcal{D} as follows: $\mathcal{T}_{\Pi_{\mathcal{D}}(P)}(\mathcal{I}) = \mathcal{J}$, where $\mathcal{J} = (\mathcal{D}, \mathcal{R})$ and \mathcal{R} is the following set of terms:*

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

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 [19] that $\mathcal{T}_{\Pi_{\mathcal{D}}(P)}$ is monotonic and so has a *unique* least fixpoint, denoted $\text{lfp}(\mathcal{T}_{\Pi_{\mathcal{D}}(P)})$.

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

Of course, a program can have different initial models, since there can be different anonymous domains and different skolemizations. However, all initial models are isomorphic.

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

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

To answer queries and to reason about logical statements we need to define the notion of *logical entailment* of one scoped formula by another. The definition is identical to the one used in classical logic.

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

This definition has 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

[10]. However, the difference is that the entailment relationship here does not hold between a *proper instance*⁴ of an RDF graph and the graph itself. For instance, $\{ \text{john}[\text{likes} \rightarrow \text{food}] \}$ is a proper instance of both $\{ _ \# [\text{likes} \rightarrow \text{food}] \}$ and $\{ \text{john}[\text{likes} \rightarrow _ \#] \}$, but

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

Indeed, when applied to RDF graphs, the notion of entailment defined above 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.

In contrast, the notion of entailment defined in the proposed RDF model theory [10] 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 can be easily captured in our framework. Namely, we can define the following notion of entailment, denoted \approx .

Definition 13 (Embedding). *Let P and Q be two F-logic programs over a language \mathcal{L} , \mathcal{C} be the set of constants in \mathcal{L} , \mathcal{D}_1 and \mathcal{D}_2 be two anonymous domains, and $\mathcal{I} = (\mathcal{D}_1, \mathcal{R})$ and $\mathcal{J} = (\mathcal{D}_2, \mathcal{S})$ be the initial models of P and Q , respectively. We say that Q can be embedded into P , denoted $P \approx Q$, iff there is an augmented domain mapping $\lambda: \mathcal{D}_2 \rightarrow \mathcal{D}_1 \cup \mathcal{C}$ such that $\lambda(\mathcal{S}) \subseteq \mathcal{R}$.*

Note that in Definition 13 the choices of the initial models for P and Q are not important, since by Corollary 2 all initial models of an F-logic program are isomorphic.

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

Theorem 2. *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 $P \approx Q$.*

Revisiting the previous example where $\{ \text{john}[\text{likes} \rightarrow \text{food}] \}$ is a proper instance of both $\{ _ \# [\text{likes} \rightarrow \text{food}] \}$ and $\{ \text{john}[\text{likes} \rightarrow _ \#] \}$, we have

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

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

⁴ A proper instance of an RDF graph is obtained by replacing one or more anonymous resources with arbitrary named resources [10].

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, even if $P \cong Q$, 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.

It turns out, however, that if Q is a formula that does not use anonymous ID symbols then the two notions of entailment are the same.

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

5 Compositionality of Semantics

It is our contention that a logic language for the Web should have the *compositionality* property. Intuitively, this means that the result of putting together, say, two RDF documents should be another valid RDF document. Intuitively, from the point of view of semantics this means that 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 and the current proposal for the RDF model theory [10] 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 4. *Let P_1 and P_2 be two F-logic programs, \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). Suppose also that P_1 and P_2 use disjoint vocabularies, i.e., the sets of named constants that occur in P_1 and P_2 are disjoint. Then $\mathcal{I}_1 \uplus \mathcal{I}_2$ is an initial model of $P_1 \cup P_2$.*

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 the RDF triple syntax:

{loves, [X], [Mary]}

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

{inventor, [X], [Bulb]}

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

{loves, [X], [Mary]}
{inventor, [X], [Bulb]}

Although the interpretation { {loves, [John], [Mary]} } is a model for the document P_1 and { {inventor, [ThomasEdison], [Bulb]} } is a model for P_2 , their (disjoint) union, { {loves, [John], [Mary]}, {inventor, [ThomasEdison], [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.

6 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 [10].

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. 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, the objects referred to via X and Y would have nothing to do with each other. For instance, John believes 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 state

```

('http://foo.org/TheBulb'[inventor  $\rightarrow$  'http://foo.org/ThomasEdison'])
  [veracity  $\rightarrow$  true, authority  $\rightarrow$  'http://www.britannica.com/'].

```

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

```

Statement  $\leftarrow$  Statement[veracity  $\rightarrow$  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[believes  $\rightarrow$  S1], X[believes  $\rightarrow$  S2]  $\leftarrow$  X[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

```

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

```

But (john[likes \rightarrow sally]) [statementAbout \rightarrow friendship] is not necessarily a true statement.

An important advantage of our semantics is that it is developed in a general framework, which enables reasoning about reified statements — not only 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 third party.* This knowledge can be encoded using the following rule:

```

'http://xyz.com/Bob'[believes  $\rightarrow$  Statement[authority  $\rightarrow$  A]]  $\leftarrow$ 
  'http://xyz.com/Alice'[says  $\rightarrow$  Statement[authority  $\rightarrow$  A]].

```

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

7 Conclusion

In this paper we presented an extension of F-logic that supports anonymous objects 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 semantics and a proof theory and has been implemented in the \mathcal{F} LORA-2 system [24]. In particular, our semantics rectifies a number of drawbacks of the current proposal for the RDF model theory [10]: 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 [10], 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 object identity 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 object identities are related to the so-called *pure values* in object-oriented databases [1, 17] 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 in \mathcal{F} LORA-2 to provide a clean semantics (and an implementation) for aggregate operators in \mathcal{F} LORA-2, which take a query (*i.e.*, a reified formula) as an argument and perform summarization operations over the set of answers to the query.

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. A.J. Bonner and M. Kifer. An overview of transaction logic. *Theoretical Computer Science*, 133:205–265, October 1994.
3. 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.
4. 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. <http://www.aidministrator.nl/publications/otk-del8.pdf>.
5. 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.
6. 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.
7. 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.
8. 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.

9. 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.
10. P. Hayes (editor). RDF Model Theory. Technical report, W3C, April 2002. <http://www.w3.org/TR/rdf-nt/>.
11. 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.
12. 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.
13. 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>.
14. 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.
15. 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.
16. 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.
17. M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.
18. 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/>.
19. J. W. Lloyd. *Foundations of Logic Programming*. Springer Verlag, 1984.
20. D. Perlis. Languages with self-reference i: Foundations. *Artificial Intelligence*, 25:301–322, 1985.
21. M. Sintek and S. Decker. TRIPLE – An RDF query, inference, and transformation language. In *Deductive Databases and Knowledge Management (DDL’2001)*, October 2001.
22. 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.
23. 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.
24. G. Yang and M. Kifer. Flora-2: User’s manual. <http://flora.sourceforge.net/>, March 2002.
25. G. Yang and M. Kifer. Well-founded optimism: Inheritance in frame-based knowledge bases. Submitted for publication, 2002.