Rules and Ontologies in F-logic *

Michael Kifer
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA

Appeared in Reasoning Web, Springer LNCS 3565, 2005.
`http://www.springerlink.com/content/1j9cprv5pl55h2nb/`

Abstract

F-logic is a formalism that integrates logic with object-oriented programming in a clean and declarative fashion. It has been successfully used for information integration, ontology modeling, agent-based systems, software engineering, and more. This paper gives a brief overview of F-logic and discusses its features from the point of view of an ontology language.

1 Introduction

F-logic [15] extends classical predicate calculus with the concepts of objects, classes, and types, which are adapted from object-oriented programming. In this way, F-logic integrates the paradigms of logic programming and deductive databases with the object-oriented programming paradigm.

Most of the applications of F-logic have been as a language for intelligent information systems based on the logic programming paradigm. This was the original motivation for the development of F-logic. More recently, F-logic has been used to represent ontologies and other forms of Semantic Web reasoning [9, 8, 25, 1, 23, 14].

Currently several implementations of the rule-based subset of F-logic are available. Ontobroker [20] is a commercial F-logic based engine developed by Ontoprise. It is designed as a knowledge-base component for a Java application. Flora-2 [31] is an open-source system that was developed at Stony

*This work was supported in part by NSF grant CCR-0311512 and by U.S. Army Medical Research Institute under a subcontract through Brookhaven National Lab.

Brook as part of a research project. Unlike Ontobroker which is designed to serve Java applications, Flora-2 is a complete programming environment for developing knowledge-intensive applications. It integrates F-logic with other novel formalisms such as HiLog [6] and Transaction Logic [5]. TRIPLE [24] is a partial implementation of F-logic with a particular emphasis on interoperability with RDF. Older, unmaintained F-logic based systems are also available, such as SILRI[1] and FLORID.[2]

In this section we first survey the main features of F-logic and then discuss its use as an ontology language.

2 Overview of F-logic

F-logic extends and subsumes predicate calculus both syntactically and semantically. In particular, it has a monotonic logical entailment relationship, and its proof theory is sound and complete with respect to the semantics. F-logic comes in two flavors: the first-order flavor and the logic programming flavor. The first-order flavor of F-logic can be viewed as a syntactic variant of classical logic, which makes an implementation through source-level translation possible [15, 27, 31]. The logic programming flavor uses a subset of the syntax of F-logic, but gives it a different, non-first-order semantics.

To understand the relationship between the first-order variant of F-logic and its logic programming variant, recall that standard logic programming [18] is built on top of the rule-based subset of the classical predicate calculus by adding non-monotonic extensions. By analogy, object-oriented logic programming is constructed based on the rule-based subset of F-logic by adding the appropriate non-monotonic extensions [32, 31, 20]. These extensions are intended to capture the semantics of negation-as-failure, like in standard logic programming [26], and the semantics of multiple inheritance with overriding (which does not arise in the standard case).

2.1 Basic Syntax

F-logic uses first-order variable-free terms to represent *object identity* (abbr., OID); for instance, john and father(mary) are possible Ids of objects. Objects can have single-valued or set-valued attributes. For instance,

$$\begin{array}{l}\mathsf{mary}[\mathsf{spouse} \rightarrow \mathsf{john},\ \mathsf{children} \twoheadrightarrow \{\mathsf{alice},\mathsf{nancy}\}].\\ \mathsf{mary}[\mathsf{children} \twoheadrightarrow \{\mathsf{jack}\}].\end{array}$$

[1]http://ontobroker.semanticweb.org/silri/

[2]http://www.informatik.uni-freiburg.de/ dbis/florid/

Such formulas are called F-logic *molecules*. The first formula says that object mary has an attribute spouse, which is single-valued and whose value is the OID john. It also says that the attribute children is set-valued and its value is a set that *contains* two OIDs: alice and nancy. We emphasize "contains" because sets do not need to be specified all at once. For instance, the second formula above says that mary has an additional child, jack.

While some attributes of an object are specified explicitly, as facts, other attributes can be defined using deductive rules. For instance, we can derive $\mathsf{john[children} \twoheadrightarrow \{\mathsf{alice, nancy, jack}\}]$ using the following deductive rule:

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

Here we adopt the standard convention in logic programming that uppercase symbols denote variables while symbols beginning with a lowercase letter denote constants.

F-logic objects can also have *methods*, which are functions that take arguments. For instance,

$$\mathsf{john[grade(cs305,fall2004)} \rightarrow \mathsf{100,\ courses(fall2004)} \twoheadrightarrow \{\mathsf{cs305,cs306}\}].$$

says that john has a single-valued method, grade, whose value on the arguments cs305 (a course identifier) and fall2004 (a semester designation) is 100; it also has a set-valued method courses, whose value on the argument fall2004 is a set of OIDs that contains course identifiers cs305 and cs306. Like attributes, methods can be defined using deductive rules.

The F-logic syntax for *class membership* is john:student and for *subclass relationship* it is student::person. Classes are treated as objects and it is possible for the same object to play the role of a class in one formula and of an object in another. For instance, in the formula student:class, the symbol student plays the role of an object, while in student::person it appears in the role of a class.

In addition, F-logic provides the means for specifying schema information through *signature* formulas. For instance, $\mathsf{person[name} \Rightarrow \mathsf{string,\ child} \Rrightarrow \mathsf{person]}$ is a signature formula that says that class person has two attributes: a single-valued attribute name and a set-valued attribute child. It further says that, the first attribute returns objects of type string and the second returns sets of objects such that each object in the set is of type person. F-logic also supports first-order predicate syntax and in this way it extends classical predicate calculus and integrates the relational and object-oriented paradigms in knowledge representation.

We remark that attempts are being made to unify the syntax of the various implementations of F-logic, such as Ontobroker [20] and Flora-2 [31].

Among the more significant forthcoming changes (as far as this overview goes) are that all attributes will be treated as set-valued (for which $\rightarrow$ will be used instead of $\twoheadrightarrow$). To capture the single-valued attributes of old, cardinality constraints will be introduced. The syntax of variables will also change: instead of capitalization, all variables will be prefixed with the "?" sign.

2.2 Querying Meta-Information

F-logic provides simple and natural means for exploring the structure of object data. Both schema information associated with classes and the structure of individual objects can be queried by simply putting variables in the appropriate syntactic positions. For instance, to find the set-valued methods that are defined in the *schema* of class student and return objects of type person, one can ask the following query:

$$?\text{-}\quad \mathsf{student}[\mathsf{M} \Rrightarrow \mathsf{person}].$$

The next query is about the type of the results of the attribute name in class student. In addition, the query returns all the superclasses of class student.

$$?\text{-}\quad \mathsf{student::C\ and\ student}\,[\mathsf{name} \Rightarrow \mathsf{T}].$$

The above queries are *schema-level meta-queries* because they involve the subclass relationship and the type information (as indicated by the operators ::, $\Rightarrow$, and $\Rrightarrow$). In contrast, the following queries involve object data (rather than schema); they return the methods that have a known value for the object with the OID john:

$$?\text{-}\quad \mathsf{john}[\mathsf{SingleM} \rightarrow \mathsf{SomeValue}].$$
$$?\text{-}\quad \mathsf{john}[\mathsf{SetM} \twoheadrightarrow \mathsf{SomeValue}].$$

Like the previous queries, the last two deal with meta-information about objects, but they examine object data rather than schema. Therefore, they are called *instance-level meta-queries.* The two kinds of meta-queries can return different results for several reasons. First, in case of semistructured data, schema information might be incomplete, so additional attributes might be defined for individual objects but not mentioned in the schema. Second, even if the schema is complete, the values of some attributes can be undefined for some objects. In this case, the undefined attributes will not be returned by instance-level meta-queries, but they would be returned by schema-level meta-queries.

2.3 Path Expressions

In addition to the basic syntax, F-logic supports so-called *path expressions*, which generalize the dot-notation in object-oriented programming languages such as Java or C++. Path expressions simplify navigation along attribute and method invocations, and help avoid explicit join conditions [10].

A *single-valued* path expression, O.M, refers to the *unique* object R for which $\mathsf{O[M \rightarrow R]}$ holds; a *set-valued* path expression, O..M, refers to some object, R, such that $\mathsf{O[M \twoheadrightarrow \{R\}]}$ holds. Here the symbols O and M can be either OIDs or other path expressions. Furthermore, M can be a method with arguments. For instance, $\mathsf{O.M(P_1,\ldots,P_k)}$ is a valid path expression that refers to the object R that satisfies $\mathsf{O[M(P_1,\ldots,P_k) \rightarrow R]}$.

Since path expressions can occur anywhere an OID is allowed, they can be nested within other F-logic molecules and provide alternative and much more concise ways of addressing objects in a knowledge base. For instance, the path expression

$$\mathsf{Paper[authors \twoheadrightarrow \{Author[name \rightarrow john]\}].publication..editors}$$

refers to all editors of those papers in which john is the name of a coauthor. An equivalent representation in terms of the basic F-logic syntax would be

$$\begin{array}{l}\mathsf{Paper[authors \twoheadrightarrow Author]\ \ and\ \ Author[name \rightarrow john]\ \ and} \\ \qquad\quad \mathsf{Paper[publication \rightarrow P]\ \ and\ \ P[editor \twoheadrightarrow E]}\end{array}$$

The reader has probably noticed the conceptual similarity between the path expressions in F-logic, introduced in [10], and the language of XPath, which was developed later but with a similar purpose in mind.

2.4 Additional Features

F-logic includes a number of other language constructs that can be very useful in knowledge representation in general and on the Semantic Web in particular. One of these important features is the equality predicate, :=:, which can be used to declare two objects to be the same. For instance, mary :=: mother(john) asserts that the object with the OID mary and the object with the OID mother(john) are one and the same object. The presence of explicit equality goes against the grain of standard logic programming, which assumes a particular built-in theory of equality, where two variable-free terms are equal if and only if they are identical. A common use of explicit equality on the Semantic Web is to provide assertions stating that a pair of syntactically different URIs refer to the same document.

Another important feature of some of the F-logic implementations, such as Flora-2, is integration with HiLog [6]. This allows a higher degree of meta-programming in a clean and logical way. For instance, one can ask a query of the form

$\mathsf{?\text{-}\ person[M(Arg)} \Rightarrow \mathsf{person]}.$

and obtain a set of all methods that take one argument, are declared to be part of the schema of class person, and return results that are objects belonging to class person. Note that M(Arg) is not a first-order term, since it has a variable in the position of a function symbol; such terms are not allowed in Prolog-based logic programming languages.

Later additions to F-logic include reification and anonymous object identity [30, 14]. Both features are deemed to be important for Semantic Web and are included in RDF [16, 13]. It has been argued, however, that the RDF formalization of these notions is less that optimal and that the proposal requires significant extensions in order to be useful for advanced applications [30]. A convincing use of the extensions provided by F-logic has been given in [14] in the context of Semantic Web Services.

2.5 Inheritance

F-logic supports both *structural* and *behavioral* inheritance. The former refers to inheritance of method types from superclasses to their subclasses and the latter deals with inheritance of method definitions from superclasses to subclasses.

Structural inheritance is defined by very simple inference rules:

If $\mathsf{subcl::cl}$, $\mathsf{cl[attr} \star\!\!\Rightarrow \mathsf{type]}$ then $\mathsf{subcl[attr} \star\!\!\Rightarrow \mathsf{type]}$
If $\mathsf{obj:cl}$, $\mathsf{cl[attr} \star\!\!\Rightarrow \mathsf{type]}$ then $\mathsf{obj[attr} \Rightarrow \mathsf{type]}$

Similar rules hold for multi-valued attributes that are designated using the arrows $\star\!\!\Rightarrow\!\!\!\!\Rightarrow$ and $\Rightarrow\!\!\!\!\Rightarrow$. The statement $\mathsf{cl[attr} \star\!\!\Rightarrow \mathsf{type]}$ in the the above rules states that attr is an *inheritable* attribute, which means that both its type and value are inheritable by the subclasses and members of class cl. Inheritability of the type of an attribute is indicated with the star attached to the symbol $\star\!\!\Rightarrow$. In all previous examples we have been dealing with *non-inheritable* attributes, which were designated with star-less arrows. Note that when the type of an attribute is inherited to a subclass it remains inheritable. However, when it is inherited to a member of the class it is no longer inheritable.

Type inheritance is not overridable; instead all types accumulate. For instance, from

```
faculty::employee.
manager::employee.
john:faculy.
faculty[reportsTo ⋆⇒ faculty].
employee[reportsTo ⋆⇒ manager].
```

we can derive two statements by inheritance: john[reportsTo ⋆⇒ faculty] and john[reportsTo ⋆⇒ manager]. The type expression for the more specific superclass, faculty, does not override the type expression for the less specific class, employee. The intended interpretation is that whoever john reports to must be both a manager and an employee. These two statements can be replaced with a single statement of the form john[reportsTo ⋆⇒ (faculty and manager)].

Behavioral inheritance is more complex. To get a flavour of behavioral inheritance, consider the following small knowledge base:

```
royalElephant::elephant.
clyde:royalElephant.
elephant[color ⋆→ grey].
royalElephant[color ⋆→ white].
```

Like with type definitions, a star attached to the arrow, $\star\!\!\rightarrow$, designates an inheritable method. For instance, color is an inheritable attribute in classes elephant and royalElephant. The inference rule that guides behavioral inheritance can informally be stated as follows. If obj is an object and cl is a class, then

obj:cl, cl[attr ⋆→ value]   should imply   obj[attr → value]

unless the inheritance is overwritten by a more specific class. The meaning of the exception here is that the knowledge base should not imply the formula obj[attr → value] if there is an intermediate class, cl', which overrides the inheritance, i.e., if obj:cl', cl'::cl are true and cl'[attr ⋆→ value'] is defined explicitly.[3] A similar exception exists in case of multiple inheritance conflicts. Note that inheritable attributes become non-inheritable after they are inherited by class members. In the above case, inheritance of the grey color is overwritten by the white color and so clyde[color → white] is derived by the rule of inheritance.

[3]The notion of an explicit definition seems obvious at first but, in fact, is quite subtle [28].

2.6 Semantics

The semantics of F-logic is based on the notion of F-structures, which extend the notion of semantic structures in classical predicate calculus. OIDs are interpreted in F-structures as elements of the domain and methods (and attributes) are interpreted as partial functions of suitable arities. The first argument of each such function is the Id of the object in whose context the method or the attribute is defined. Signature formulas are interpreted by functions whose properties are made to fit the common properties of types. Details of F-structures can be found in [15].

Armed with the notion of the F-structures, a first-order entailment relation is defined in a standard way: $\phi \models \psi$ if and only if every F-structure that satisfies ϕ also satisfies ψ. This entailment together with the sound and complete resolution-based proof theory [15] are the basis of the first-order variant of F-logic.

The semantics of the logic programming variant of F-logic is built by analogy with the corresponding development in deductive databases. The meaning of negation is made non-monotonic and is based on an extension of the well-founded semantics [26]. The interesting and nontrivial aspect of this extension is not due to negation (negation is handled analogously to [26]) but due to behavioral inheritance with overriding. Earlier we have seen an informal account of inference by inheritance. Although the rules of such inference seem natural, they present subtle problems when behavioral inheritance is used together with deductive rules. To understand the problem, consider the following example.

```
cl[attr *→ v1].
subcl::cl.
obj:subcl.
subcl[attr *→ v2]   :−  obj[attr → v1].
```

If we apply the rule of inheritance to this knowledge base, then obj[attr → v1] should be inherited, since no overriding takes place. However, once obj[attr → v1] is derived by inheritance, subcl[attr *→ v2] can be derived by deduction, and now we have a chicken-and-egg problem. Since subcl is a more specific superclass of obj, the derivation of subcl[attr *→ v2] appears to override the earlier inheritance of obj[attr → v1]. But this, in turn, undermines the reason for deriving subcl[attr *→ v2]. The above is only one of several suspicious derivation patterns that arise due to interaction of inheritance and deduction. The original solution reported in [15] was not model-theoretic and was problematic in several other respects as well. A satisfactory and completely model-theoretic solution was proposed in [28, 29].

3 F-logic as an Ontology Language

From the beginning, F-logic has been viewed as a natural candidate for an ontology language due to its direct support for object-oriented concepts, its frame-based syntax, and extensive support for meta-programming [9, 8, 25]. More recently it has been adopted as a basis for ontology languages for Semantic Web Services [23, 4].

3.1 The Basic Techniques

A typical ontology includes three main components:

1. *A taxonomy of classes.* This includes the specification of the class hierarchy, i.e., which classes are subclasses of other classes.

2. *Definitions of concepts.* These definitions specify the allowed attributes of each class, their types, and other properties (like symmetry or transitivity).

3. *Definitions of instances.* Instances (i.e., concrete data objects) are defined by indicating which concepts (i.e., classes) they belong to and by specifying concrete values for the attributes of those instances. Sometimes the values might not be given explicitly, but only their existence can be asserted with various degrees of precision. For instance, $\exists \mathtt{F}\, \mathtt{john}[\mathtt{father} \rightarrow \mathtt{F}]$ or $\mathtt{john}[\mathtt{father} \rightarrow \mathtt{bob}] \vee \mathtt{john}[\mathtt{father} \rightarrow \mathtt{bill}]$. Some concepts may not have explicitly defined instances. Instead, their instances may be defined by deductive rules. Such concepts are akin to database views.

In F-logic, class taxonomies are represented directly using the subclass relationship ::. Concept definitions are represented using signature formulas, such as $\mathsf{person}[\mathsf{name} \star\!\!\Rightarrow \mathsf{string},\ \mathsf{spouse} \star\!\!\Rightarrow \mathsf{person}]$. Special properties of certain attributes can be expressed using rules. For instance, to state that spouse is a symmetric relationship in class person one can write

$$\mathsf{X}[\mathsf{spouse} \rightarrow \mathsf{Y}] \quad :- \quad \mathsf{Y{:}person}\ \ \mathtt{and}\ \ \mathsf{Y}[\mathsf{spouse} \rightarrow \mathsf{X}].$$

Finally, instance definitions can be specified as facts using data molecules as follows:

```
john:student.
john[name → John, address → '123 Main St.', spouse → Mary].
```

Derived classes can be defined using rules. For instance, if the concepts of student and employee are already defined, we can define a new concept, workstudy using the following statements:

```
X:workstudy  :− X:(student and employee) and X[jobtype → J] and J:clerical.
```

Properties can also be defined using rules. For instance, if the properties mother and father are already defined, we can define the properties of parent and ancestor as follows:

```
X[parent → P]    :− X[mother → P].
X[parent → P]    :− X[father → P].
X[ancestor → A]  :− X[parent → A].
X[ancestor → A]  :− X[parent → P] and P[ancestor → A].
```

Various implementations of F-logic introduced several forms of more concise syntax. For instance, the workstudy rule above can be written as

```
X:workstudy  :− X[jobtype → J:clerical]:(student and employee).
```

the two parent rules can be abbreviated to

```
X[parent → P]  :− X[mother → P or father → P]].
```

and the second ancestor rule can be written as

```
X[ancestor → A]  :− X[parent → P[ancestor → A]].
```

3.2 Relationship to Description Logics

No discussion of F-logic is complete without a comparison with description logics (abbr. DL) [2] and, in particular, with languages such as OWL [21]. Since the first-order flavor of F-logic is an extension of classical predicate logic, it is clear that a description logic subset can be defined within F-logic and, indeed, this has been done [3]. In this sense, F-logic subsumes DLs. However, as mentioned earlier, most applications of F-logic (and all implementations known to us) use the logic programming flavor of the logic so a proper comparison would be made with that flavor.

Unlike DLs, F-logic is computationally complete. This can be a blessing or a curse depending on how one looks at this matter. On one hand, the expressive power of F-logic provides for a simple and clear specification of many problems that are beyond the expressive power of any DL. On the other hand, expressive F-logic knowledge bases provide no computational guarantees. However, many workers in the field dismiss this problem as a non-issue for two reasons:

- The exponential complexity of many problems in description logics provides very little comfort in practice, especially in reasoning with large ontologies.
- A vast class of computational problems in F-logic is decidable and has polynomial complexity. This includes all queries to knowledge bases that do not use function symbols and includes a large subclass of queries that are beyond the expressive power of DLs. Furthermore, research in logic programming and deductive database has identified large classes of knowledge bases *with* function symbols where query answering is decidable (for instance, [17]).

Nevertheless, there are two aspects where DLs provide more flexibility. First, DLs allow the user to represent existential information. For instance, one can say that there is a person with certain properties without specifying any concrete instance of such a person. In F-logic one can express only an approximation of such a statement using Skolem functions. Similarly, DLs admit disjunctive information into the knowledge base. For instance, one can say that John has a book or a bicycle. The corresponding statement in F-logic is only an approximation:

john[has → _#:(book or bicycle)].

The symbol _# here denotes a unique Skolem constant that does not occur anywhere else in the knowledge base. While this may be an acceptable approximation in some cases, it is still significantly weaker that the corresponding DL statement.

For instance, if upon closer examination it becomes known that John does not have a book, then in DLs we would conclude that John has a bicycle. In the logic programming flavor of F-logic (as in other logic programming systems) we cannot even state that John has no books directly—one has to employ some rather complex tricks. Some extensions of standard logic programming support *explicit negation* and thus can make negative information easier to specify. For instance, this problem could be overcome by combining F-logic with Courteous Logic Programming [11, 12]. Other extensions allow disjunctive information in the rule heads [22, 19], which permits statements like `john[father → bob]` ∨ `john[father → bill]`.

3.3 Example: An OWL-S Profile

We now give a more extensive example of an ontology specified using F-logic—part of an OWL-S profile [7]. OWL-S is an OWL-based Web ontology, which is intended to provide Web service providers with a core set of constructs for describing the properties and capabilities of their Web services. OWL-S often refers to externally defined data types using the namespace notation. Although some implementation of F-logic support URIs and namespaces, our example will omit all namespace definitions and will reference the corresponding external data types and concepts by enclosing them in single quotes, e.g., `'xsd:string'`.

```
'service:ServiceProfile' : 'owl:Class'.
'Profile' :: 'service:ServiceProfile'
'Profile'[
   serviceName *=> 'xsd:string',
   textDescription *=> 'xsd:string',
   'rdfs:comment'*->'Definition of Profile',
   contactInformation *=>> 'Actor',
   hasProcess *=> 'process:Process',
   serviceCategory *=>> ServiceCategory,
   serviceParameter *=>> ServiceParameter,
   hasParameter *=>> 'process:Parameter',
   hasInput *=>> 'process:Input',
   hasOutput *=>> 'process:ConditionalOutput',
   hasPrecondition *=>> 'expr:Condition',
   hasEffect *=>> 'process:ConditionalEffect'
].

hasInput[subpropertyof ->> hasParameter].
hasOutput[subPropertyOf ->> hasParameter].

// Definition of subPropertyOf
Obj[P ->> Val] :- S[subPropertyOf ->> P] and Obj[S ->> Val].

'ServiceCategory' : 'owl:Class'.
'ServiceCategory'[
   categoryName *=> 'xsd:string',
   taxonomy *=> 'xsd:string',
   value *=> 'xsd:string',
   code *=> 'xsd:string'
].

'ServiceParameter' : 'owl:Class'.
'ServiceParameter'[
    serviceParameterName *=> 'xsd:string',
    sParameter *=> 'owl:Thing'
].

'Actor' : 'owl:Class'.
'process:Process' : 'owl:Class'.
'expr:Condition' : 'owl:Class'.
'process:Input' : 'owl:Class'.
'process:ConditionalOutput' : 'owl:Class'.
'process:ConditionalEffect' : 'owl:Class'.
'process:Parameter' : 'owl:Class'.
```

The above ontology is fairly simple. The frame-based syntax of F-logic enables concise and clear description of the properties of the various classes defined by OWL-S. The only place where a more sophisticated aspect of F-logic is necessary is the definition of `subPropertyOf`, a property that applies to attributes when they are considered as objects in their own right. To define the meaning of this property we use an F-logic rule.

OWL distinguishes between *object properties* and *data type properties*, and defines two OWL classes for that. The class 'owl:ObjectProperty' is populated by object properties, which are attributes whose range is an OWL class. The class 'owl:DataTypeProperty' is populated by data type properties, which are defined as attributes whose range is an XML type, such as 'xsd:string'. In the OWL-based OWL-S ontology, every property must be explicitly specified to be in either the 'owl:ObjectProperty' class or the 'owl:DataTypeProperty' class. In F-logic this can be done much more elegantly using rules:

$$\begin{aligned}
&\mathsf{Prop} : \mathsf{property(Range)}\ \ :-\ \ \mathsf{Domain[Prop} \star\!\!\Rightarrow \mathsf{Range}\ \ \mathsf{or}\ \ \mathsf{Prop} \star\!\!\Rightarrow\!\!\!\!\Rightarrow \mathsf{Range]}.\\
&\mathsf{Prop} : \mathsf{'owl{:}ObjectProperty'}\ \ :-\\
&\qquad\qquad \mathsf{Prop} : \mathsf{property(Range)}\ \ \mathsf{and}\ \ \mathsf{Range} : \mathsf{'owl{:}Class'}.\\
&\mathsf{Prop} : \mathsf{'owl{:}DataTypeProperty'}\ \ :-\\
&\qquad\qquad \mathsf{Prop} : \mathsf{property(Range)}\ \ \mathsf{and}\ \ \mathsf{not\ Range} : \mathsf{'owl{:}Class'}.
\end{aligned}$$

This example provides a glimpse on how the ability of F-logic to operate at the meta-level provides significant benefits in terms of conciseness and readability of ontology specifications.

References

[1] J. Angele and G. Lausen. Ontologies in F-logic. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*, pages 29–50. Springer Verlag, Berlin, Germany, 2004.

[2] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook.* Cambridge University Press, 2002.

[3] M. Balaban. The f-logic approach for description languages. *Annals of Mathematics and Artificial Intelligence*, 15(1):19–60, 1995.

[4] D. Berardi, B. Grosof, M. Gruninger, R. Hull, M. Kifer, D. Martin, S. McIlraith, and J. Su. Semantic web services language. Technical report, Semantic Web Services Initiative, February 2005. http://www.daml.org/services/swsl/, in preparation.

[5] A. 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] W. Chen, M. Kifer, and D. Warren. HiLog: A foundation for higher-order logic programming. *Journal of Logic Programming*, 15(3):187–230, February 1993.

[7] O.-S. Coalition. Owl-s. http://www.daml.org/services/owl-s/1.1/, December 2004.

[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] D. Fensel, M. Erdmann, and R. Studer. Ontobroker: How to make the WWW intelligent. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, Banff, Canada, 1998.

[10] J. Frohn, G. Lausen, and H. Uphoff. Access to objects by path expressions and rules. In *Proceddings of the Intl. Conference on Very Large Databases*, pages 273–284, Santiago, Chile, 1994. Morgan Kaufmann, San Francisco, CA.

[11] B. Grosof. Prioritized conflict handling for logic programs. In *International Logic Programming Symposium*, pages 197–211, 1997.

[12] B. Grosof. A courteous compiler from generalized courteous logic programs to ordinary logic programs. Technical Report RC 21472, IBM, July 1999.

[13] P. Hayes. RDF model theory. W3C Working Draft, 10 October 2003. Available at `http://www.w3.org/TR/rdf-mt/`.

[14] M. Kifer, R. Lara, A. Polleres, and C. Zhao. A logical framework for web service discovery. In *Semantic Web Services Workshop*, November 2004.

[15] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[16] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax. W3C Working Draft, 10 October 2003. Available at `http://www.w3.org/TR/rdf-concepts/`.

[17] N. Lindenstrauss and Y. Sagiv. Automatic termination analysis of logic programs. In *International Conference on Logic Programming*, 1997.

[18] J. W. Lloyd. *Foundations of logic programming (second, extended edition)*. Springer series in symbolic computation. Springer-Verlag, New York, 1987.

[19] J. Lobo, J. Minker, and A. Rajasekar. *Foundations of Disjunctive Logic Programming*. MIT Press, Cambridge, Massachusetts, 1992.

[20] Ontoprise, GmbH. Ontobroker manual. http://www.ontoprise.com/.

[21] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. W3C Candidate Recommendation, 18 August 2003. Available at `http://www.w3.org/TR/owl-semantics/`.

[22] T. Przymusinski. Well-founded and stationary models of logic programs. *Annals of Mathematics and Artificial Intelligence*, 12:141–187, 1994.

[23] D. Roman, H. Lausen, and U. Keller. Web services modeling ontology. Technical report, DERI, November 2004. http://www.wsmo.org/2004/d2/.

[24] M. Sintek, S. Decker, and A. Harth. The triple system. http://triple.semanticweb.org/, 2003.

[25] S. Staab and A. Maedche. Knowledge portals: Ontologies at work. *The AI Magazine*, 22(2):63–75, 2000.

[26] A. Van Gelder, K. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *J. ACM*, 38(3):620–650, July 1991.

[27] 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.

[28] G. Yang and M. Kifer. Well-founded optimism: Inheritance in frame-based knowledge bases. In *Intl. Conference on Ontologies, DataBases, and Applications of Semantics for Large Scale Information Systems (ODBASE)*, October 2002.

[29] G. Yang and M. Kifer. Inheritance and rules in object-oriented semantic web languages. In *Rules and Rule Markup Languages for the Semantic Web (RuleML03)*, volume 2876 of *Lecture Notes in Computer Science.* Springer Verlag, November 2003.

[30] G. Yang and M. Kifer. Reasoning about anonymous resources and meta statements on the semantic web. *Journal on Data Semantics, LNCS 2800*, 1:69–98, September 2003.

[31] G. Yang, M. Kifer, and C. Zhao. Flora-2: User's Manual. http://flora.sourceforge.net/, June 2002.

[32] G. Yang, M. Kifer, and C. Zhao. Flora-2: A rule-based knowledge representation and inference infrastructure for the semantic web. In *International Conference on Ontologies, Databases and Applications of Semantics (ODBASE-2003)*, November 2003.