



D16.2v0.2. F-logic/XML - An XML Syntax for F-logic

WSMO Working Draft 24 March 2004

This version:

<http://www.wsmo.org/2004/d16/d16.2/v0.1/20040324/>

Latest version:

<http://www.wsmo.org/2004/d16/d16.2/v0.2/>

Previous version:

<http://www.wsmo.org/2004/d16/d16.2/v0.1/>

Editors:

Jos de Bruijn
Michael Kifer

This document is also available in non-normative [PDF](#) version.

Copyright © 2004 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Abstract

This document defines F-logic/XML - an XML syntax for F-logic [[Kifer et al., 1995](#)]. The proposal captures only the rule-based syntax (we believe this will suffice for WSML), but the syntax is general enough to permit disjunctions and explicit quantification.

The F-Logic/XML syntax provided in this document can be used to encode logical expressions and is used in WSMO deliverable D16.3, the XML syntax for WSML, for logical expressions.

Table of contents

- [1. Introduction](#)
- [2. XML Syntax](#)
- [3. XSLT Stylesheet for Converting F-logic/XML to Regular F-logic Syntax](#)
- [4. References](#)
- [Appendix A. XML Schema for F-Logic](#)
- [Appendix B. XSLT for Converting F-Logic/XML to F-logic](#)
- [Appendix C. Example F-Logic/XML and a Corresponding F-logic Statements](#)

1. Introduction

This document defines an XML syntax for F-logic [[Kifer et al., 1995](#)]. This syntax, henceforth referred to as F-logic/XML, captures a large and the most useful subset of F-logic, which includes rules, facts, and queries. The syntax goes far beyond Horn rules by permitting disjunction and explicit quantification.

The proposed XML syntax captures some of the HiLog extensions of F-logic [Yang & Kifer, 2004], such as variables over function symbols and reification. This version does not cover predicate symbols and some of the other useful extensions that were introduced in the [FLORA-2 system](#). These extensions will be added as needed.

The general syntax of F-logic rules captured in F-logic/XML is

forall *FREE-VARIABLES* (*HEAD* <- *BODY*)

where *FREE-VARIABLES* is a list of all free variables in the rule.

HEAD and *BODY* both have the form

CONJUNCT and ... and *CONJUNCT*

Variables that are not explicitly quantified (those that are free) in the head or the body are universally quantified outside of the clause. The free variables are not explicitly listed in the XML syntax.

CONJUNCT has the form

VARIABLE-LIST *F-logic-MOLECULE*

or

VARIABLE-LIST *DISJUNCT*

where *DISJUNCT* is of the form

VARIABLE-LIST *F-logic-MOLECULE*

or

VARIABLE-LIST *CONJUNCT*

The explicitly quantified variables in the head and the body are captured by the meta-type *VARIABLE-LIST*; they are listed as part of *CONJUNCT* and *DISJUNCT* meta types. The quantifiers are not explicitly given listed the syntax. Instead, the explicitly quantified variables in the rule head are assumed to be existential and the explicitly quantified variables in the rule body are quantified universally.

2. XML Syntax

The XML Schema (see [Appendix A](#)) captures the syntax of F-logic rules, as described earlier.

[Appendix C](#) contains an example F-Logic program encoded using the F-Logic XML syntax.

3. XSLT Stylesheet for Converting F-logic/XML to the Regular F-logic Syntax

The XSLT stylesheet (see [Appendix B](#)) transforms XML F-logic syntax back to the original, "human-readable" F-logic syntax.

4. References

[Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: Logical foundations of object oriented and frame-based languages. *Journal of the ACM*, 42(4):741-843, 1995.

[Yang & Kifer, 2004] G. Yang and M. Kifer: Reasoning about Anonymous Resources and Meta Statements on the Semantic Web. *Journal of Data Semantics*, 2004.

Appendix A. XML Schema for F-Logic/XML

```
<?xml version="1.0" encoding="iso-8859-1"?>
<schema targetNamespace="http://www.wsmo.org/2004/d16/d16.2/v0.1/" xmlns:flg="http://www.wsmo.org
  xmlns="http://www.w3.org/2001/XMLSchema" elementFormDefault="unqualified"
  attributeFormDefault="unqualified">
  <!-- version: 2004-03-13 -->
  <!-- XMLSchema for F-logic -->
  <!-- This relies on circularly defined types, which are not disallowed in
    XMLSchema -->
  <!-- Simplified syntax: only ->, *->, =>, *=>. Makes the schema simpler. -->
  <!-- Defined the schema for rules only.
    Don't think more general forms are needed.
    Rules are quite general: can have quantifiers, disjunctions, etc. -->
  <element name="flogic">
    <complexType>
      <sequence>
        <element name="rule" type="flg:ruleType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="ruleType">
    <sequence>
      <element name="head" type="flg:conjunctType" minOccurs="0" maxOccurs="unbounded"/>
      <element name="body" type="flg:conjunctType" minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="name" type="string"/>
  </complexType>
  <!-- conjunctType and disjunctType are mutually recursive. -->
  <complexType name="conjunctType">
    <sequence>
      <element name="variable" type="flg:variableType" minOccurs="0" maxOccurs="unbounded"/>
      <choice>
        <element name="molecule" type="flg:moleculeType" maxOccurs="unbounded"/>
        <element name="disjunct" type="flg:disjunctType" maxOccurs="unbounded"/>
        <!-- a conjunct is a molecule or a disjunct -->
      </choice>
      <!-- List of quantified vars: existential in the head; universal in the body.
        Unlisted vars are universally quantified outside of the rule. -->
    </sequence>
  </complexType>
  <complexType name="disjunctType">
    <sequence>
      <element name="variable" type="flg:variableType" minOccurs="0" maxOccurs="unbounded"/>
      <choice>
        <element name="molecule" type="flg:moleculeType" maxOccurs="unbounded"/>
        <element name="conjunct" type="flg:conjunctType" maxOccurs="unbounded"/>
        <!-- a disjunct is a molecule or a conjunct -->
      </choice>
      <!-- List of quantified vars: existential in the head; universal in the body.
        Unlisted vars are universally quantified outside of the clause. -->
    </sequence>
  </complexType>
  <complexType name="moleculeType">
    <sequence>
      <element name="object" type="flg:termType"/>
      <element name="superclass" minOccurs="0">
        <complexType>
          <sequence>
            <element name="class" type="flg:termType"/>
            <!-- superclass of object -->
          </sequence>
          <attribute name="isaType" type="flg:isaType"/>
          <!-- isaType: member or subclass -->
        </complexType>
      </element>
      <element name="methodSpec" type="flg:methodSpecType" minOccurs="0"/>
      <!-- Optionally specify obj:class or obj::class -->
    </sequence>
  </complexType>
</schema>
```

```

</complexType>
<complexType name="methodSpecType">
  <sequence>
    <element name="name" type="flg:termType"/>
    <element name="result">
      <complexType>
        <choice>
          <element name="oid" type="flg:termType" maxOccurs="unbounded"/>
          <element name="molecule" type="flg:moleculeType"
            maxOccurs="unbounded"/>
        </choice>
      </complexType>
    </element>
    <!-- method name & args -->
    <!-- method results -->
  </sequence>
  <attribute name="arrow" type="flg:arrowType"/>
</complexType>
<complexType name="reifiedFormulaType">
  <choice>
    <element name="molecule" type="flg:moleculeType"/>
    <element name="disjunct" type="flg:disjunctType"/>
    <element name="conjunct" type="flg:conjunctType"/>
    <!-- Allow reification of vars, as in Flora-2 (meaning no-op)? -->
  </choice>
</complexType>
<!-- *****Auxiliary types***** -->
<!-- Variable -->
<complexType name="variableType">
  <attribute name="name" type="string"/>
</complexType>
<complexType name="constantType">
  <attribute name="name" type="string"/>
</complexType>
<!-- Term is self-recursive. -->
<!-- Define HiLog term. -->
<complexType name="termType">
  <choice>
    <element name="variable" type="flg:variableType"/>
    <element name="constant" type="flg:constantType"/>
    <element name="reification" type="flg:reifiedFormulaType"/>
    <sequence>
      <element name="functor" type="flg:termType"/>
      <element name="argument" type="flg:termType" maxOccurs="unbounded"/>
    </sequence>
  </choice>
</complexType>
<simpleType name="isaType">
  <restriction base="string">
    <enumeration value=":"/>
    <enumeration value="::"/>
  </restriction>
</simpleType>
<!-- I recommend to use only single-shafted arrows.
Simplifies the schema and syntax -->
<simpleType name="arrowType">
  <restriction base="string">
    <enumeration value="->"/>
    <enumeration value="=>"/>
    <enumeration value="*->"/>
    <enumeration value="*=>"/>
  </restriction>
</simpleType>
</schema>

```

Appendix B. XSLT for Converting F-Logic/XML to the Regular F-logic Syntax

```

<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>

  <!-- XSLT document for the translation of F-Logic XML into the WSML F-Logic syntax

  creator: Jos de Bruijn
  date: 2004-03-18 -->

  <!-- source XML documents are assumed to be valid according to the schema flogic-updatesJos.xml

```

```

with date 2004-03-12 -->
<!-- destination documents should be valid according to the grammar defined in WSMO D16.1, the
version to be published 2004-03-15 -->

<!-- In the generated F-Logic syntax, the AND has precedence over OR and since both AND and OR ar
commutative, no brackets are needed -->

<!-- writes the header information -->

  <xsl:template match="/">
% F-Logic generated from F-Logic XML syntax using the WSMO XSLT
%
%

<xsl:apply-templates/>
  </xsl:template>

<!-- writes the rule itself; the building block for F-Logic statements -->
  <xsl:template match="rule">
<xsl:for-each select="head"><xsl:call-template name="conjunct"/><xsl:if
test="position()=last()"> AND </xsl:if></xsl:for-each><xsl:if
test="body"> <- </xsl:if><xsl:for-each select="body"><xsl:call-template
name="conjunct"/></xsl:if></xsl:for-each> AND </xsl:if></xsl:for-each>.
</xsl:template>
  <xsl:template match="conjunct" name="conjunct">
    <xsl:if test="variable"><xsl:if
      test="ancestor-or-self::body">FORALL </xsl:if><xsl:if
        test="ancestor-or-self::head">EXISTS </xsl:if><xsl:for-each
          select="variable"><xsl:apply-templates select="."/><xsl:if
            test="position()=last()">, </xsl:if></xsl:for-each> (</xsl:if><xsl:for-each
              select="molecule"><xsl:apply-templates select="."/><xsl:if
                test="position()=last()"> AND </xsl:if></xsl:for-each><xsl:for-each
                  select="disjunct"><xsl:apply-templates select="."/><xsl:if
                    test="position()=last()"> OR </xsl:if></xsl:for-each><xsl:if
                      test="variable"></xsl:if></xsl:template>
    <xsl:template match="functor">
      <xsl:apply-templates/>(<xsl:for-each select="../argument"><xsl:apply-templates
        select="*"></xsl:if> test="position()=last()", </xsl:if></xsl:for-each>)</xsl:templa
    <xsl:template match="superclass">
      <xsl:value-of select="@isaType"/>
      <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="methodSpec">[<xsl:apply-templates select="name"/> <xsl:value-of
      select="@arrow"/><xsl:apply-templates select="result"/>]</xsl:template>
    <xsl:template match="variable">
      <xsl:value-of select="@name"/>
    </xsl:template>
    <xsl:template match="molecule">
      <xsl:apply-templates/>
    </xsl:template>
    <xsl:template match="disjunct">
      <xsl:if test="variable"><xsl:if test="ancestor-or-self::body">FORALL </xsl:if><xsl:if
        test="ancestor-or-self::head">EXISTS </xsl:if><xsl:for-each
          select="variable"><xsl:apply-templates select="."/><xsl:if
            test="position()=last()">, </xsl:if></xsl:for-each></xsl:if><xsl:for-each
              select="molecule"><xsl:apply-templates select="."/><xsl:if
                test="position()=last()"> AND </xsl:if></xsl:for-each><xsl:for-each
                  select="conjunct"><xsl:apply-templates select="."/><xsl:if
                    test="position()=last()"> AND </xsl:if></xsl:for-each><xsl:if
                      test="variable"></xsl:if></xsl:template>
    <xsl:template match="constant">
      <xsl:value-of select="@name"/>
    </xsl:template>
    <xsl:template match="argument"></xsl:template>
  </xsl:stylesheet>

```

Appendix C. Example of F-Logic/XML and the Corresponding F-logic Statements

```

<?xml version="1.0" encoding="UTF-8"?>
<flg:flgic xmlns:flg="http://www.wsmo.org/2004/d16/d16.2/v0.1/">
  <!-- Test data to test the WSMO F-Logic XML syntax -->
  <!-- The following <rule></rule> encodes this fact (taken from the F-Logic
    JACM paper, page 7):
  bob[name -> "Bob";
    age -> 40;
    affiliation -> csl[dname -> "CS";

```

```

        mngr -> bob;
        assistents -> {john, sally}}

this encoding writes only elementary molecules
-->
<rule>
  <head>
    <molecule>
      <object>
        <constant name="bob"/>
      </object>
      <superclass isaType=":">
        <class>
          <constant name="empl"/>
        </class>
      </superclass>
      <methodSpec arrow="->">
        <name>
          <constant name="name"/>
        </name>
        <result>
          <oid>
            <constant name="Bob"/>
          </oid>
        </result>
      </methodSpec>
    </molecule>
    <molecule>
      <object>
        <constant name="bob"/>
      </object>
      <methodSpec arrow="->">
        <name>
          <constant name="age"/>
        </name>
        <result>
          <oid><constant name="40"/></oid>
        </result>
      </methodSpec>
    </molecule>
  </head>
  <head>
    <molecule>
      <object>
        <constant name="bob"/>
      </object>
      <methodSpec arrow="->">
        <name>
          <constant name="affiliation"/>
        </name>
        <result>
          <oid><constant name="cs1"/></oid>
        </result>
      </methodSpec>
    </molecule>
  </head>
  <head>
    <molecule>
      <object>
        <constant name="cs1"/>
      </object>
      <superclass isaType=":">
        <class>
          <constant name="dept"/>
        </class>
      </superclass>
      <methodSpec arrow="->">
        <name>
          <constant name="dname"/>
        </name>
        <result>
          <oid><constant name="CS"/></oid>
        </result>
      </methodSpec>
    </molecule>
  </head>
  <head>
    <molecule>
      <object>
        <constant name="cs1"/>
      </object>

```

```

        <methodSpec arrow="->">
            <name>
                <constant name="mngr"/>
            </name>
            <result>
                <oid><constant name="bob"/></oid>
            </result>
        </methodSpec>
    </molecule>
</head>
<head>
    <molecule>
        <object>
            <constant name="cs1"/>
        </object>
        <methodSpec arrow="->">
            <name>
                <constant name="assistants"/>
            </name>
            <result>
                <oid><constant name="john"/></oid>
            </result>
        </methodSpec>
    </molecule>
</head>
<head>
    <molecule>
        <object>
            <constant name="cs1"/>
        </object>
        <methodSpec arrow="->">
            <name>
                <functor>
                    <constant name="assistants"/>
                </functor>
                <argument>
                    <constant name="arg1"/>
                </argument>
                <argument>
                    <constant name="arg2"/>
                </argument>
            </name>
            <result>
                <oid><constant name="sally"/></oid>
            </result>
        </methodSpec>
    </molecule>
</head>
</rule>

<!-- The following <rule></rule> encodes this deductive rule (taken from the
F-Logic JACM paper, page 7):
E[boss -> M] <- E:empl AND D:dept AND E[affiliation -> D[mngr -> M:empl]], i.e. someones
boss is the head of that person's department
-->
<rule>
    <head>
        <molecule>
            <object>
                <variable name="E"/>
            </object>
            <methodSpec arrow="->">
                <name>
                    <constant name="boss"/>
                </name>
                <result>
                    <oid><variable name="M"/></oid>
                </result>
            </methodSpec>
        </molecule>
    </head>
    <body>
        <molecule>
            <object>
                <variable name="E"/>
            </object>
            <superclass isaType=":">
                <class>
                    <constant name="empl"/>
                </class>
            </superclass>

```

```

        </molecule>
    </body>
</body>
<body>
    <molecule>
        <object>
            <variable name="D" />
        </object>
        <superclass isaType=":">
            <class>
                <constant name="dept" />
            </class>
        </superclass>
        <methodSpec arrow="->">
            <name>
                <constant name="mngr" />
            </name>
            <result>
                <oid><variable name="M" /></oid>
            </result>
        </methodSpec>
    </molecule>
</body>
</body>
<body>
    <molecule>
        <object>
            <variable name="E" />
        </object>
        <methodSpec arrow="->">
            <name>
                <constant name="affiliation" />
            </name>
            <result>
                <oid><variable name="D" /></oid>
            </result>
        </methodSpec>
    </molecule>
</body>
</body>
<body>
    <molecule>
        <object>
            <variable name="M" />
        </object>
        <superclass isaType=":">
            <class>
                <constant name="empl" />
            </class>
        </superclass>
    </molecule>
</body>
</rule>
<!-- The following <rule></rule> encodes this deductive rule:
X:man OR X:woman <- X:person. i.e. every person is either a man or a woman
-->
<rule>
    <head>
        <variable name="X" />
        <variable name="Y" />
    <disjunct>
        <molecule>
            <object>
                <variable name="X" />
            </object>
            <superclass isaType=":">
                <class>
                    <constant name="man" />
                </class>
            </superclass>
        </molecule>
    </disjunct>
    <disjunct>
        <molecule>
            <object>
                <variable name="X" />
            </object>
            <superclass isaType=":">
                <class>
                    <constant name="woman" />
                </class>
            </superclass>
        </molecule>
    </disjunct>
</rule>

```

```

</head>
<body>
  <variable name="Y"/>
  <molecule>
    <object>
      <variable name="Y"/>
    </object>
    <superclass isaType=":">
      <class>
        <constant name="person"/>
      </class>
    </superclass>
  </molecule>
</body>
</rule>
</flg:flogic>

```

Corresponding F-logic Statements

```

% F-Logic generated from F-Logic XML syntax using the WSMO XSLT
%
%
bob:empl[name->"Bob"] AND bob[age->40] AND bob[affiliation->cs1] AND cs1:dept[dname->"CS"]
  AND cs1[mngr->bob] AND cs1[assistants->john] AND cs1[assistants(arg1, arg2)->sally].
E[boss->M] <- E:empl AND D:dept[mngr->M] AND E[affiliation->D] AND M:empl.
EXISTS X,Y (X:man OR X:woman) <- FORALL Y (Y:person).

```

Acknowledgements

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto, COG and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate programme.

The editors would like to thank to all the [members of the WSMO working group](#) for their advises and inputs to this document.

\$Date: 2004/03/24 20:38:42 \$

[webmaster](#)