XML Schema for WSML Logical Expressions

Table of Contents

top

Schema Document Properties

Target Namespace	**http://www.wsmo.org/wsml/wsml-syntax#**
Element and Attribute Namespaces	• Global element and attribute declarations belong to this schema's target namespace. • By default, local element declarations belong to this schema's target namespac • By default, local attribute declarations have no namespace.
Schema Composition	• This schema includes components from the following schema document(s): ◦ http://www.wsmo.org/TR/d16/d16.1/v0.2/xml-syntax/wsml-identifiers.x
Documentation	version: $Revision: 1.22 $ date: $Date: 2005/03/13 13:10:01 $ author: Jos de Bruijn, Re Krummenacher this schema is a module, which belongs to the WSML/XML schema specification. This schema provides WSML/XML syntax for logical expressions. This syntax for logical expressions is inspired by the RuleML FOL (First-Order Logic) variant. Necessary additions for WSML-Full were the introduction of negation-as-failure, LP implication and constraints

Declared Namespaces

Prefix	Namespace
Default namespace	**http://www.wsmo.org/wsml/wsml-syntax#**
xml	http://www.w3.org/XML/1998/namespace
xs	http://www.w3.org/2001/XMLSchema

Schema Component Representation

```
<xs:schema targetNamespace="http://www.wsmo.org/wsml/wsml-syntax#"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:include
   schemaLocation="http://www.wsmo.org/TR/d16/d16.1/v0.2/xml-syntax/wsml-identi
   ...
</xs:schema>
```

top

Global Schema Components

Element: constraint

Name	constraint
Type	complexExpressionType
Nillable	no
Abstract	no
Documentation	Database-style constraint. Corresponds with the '!-' symbol in the human readable syntax.

XML Instance Representation

```
<constraint>
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</constraint>
```

Schema Component Representation

```
<xs:element name="constraint" type=" complexExpressionType "/>
```

top

Element: impliedByLP

Name	impliedByLP
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Logic Programming rule. This rule has a head and a body. This XML element corresponds with the ':-' symbol in the human-readable syntax.

XML Instance Representation

```
<impliedByLP>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</impliedByLP>
```

Schema Component Representation

```
<xs:element name="impliedByLP" type=" binaryComplexExpressionType "/>
```

top

Complex Type: logicalExpressionType

Super-types:	None
Sub-types:	None

Name	logicalExpressionType
Abstract	no
Documentation	A WSML logical expression is either a constraint, an LP rule or a formula. The formula corresponds to an LP rule with an empty body.

XML Instance Representation

```
<...>
   Start Choice [1]
      <constraint> ... </constraint> [1]
      <impliedByLP> ... </impliedByLP> [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="logicalExpressionType">
   <xs:choice>
      <xs:element ref=" constraint "/>
      <xs:element ref=" impliedByLP "/>
```

```
      <xs:element ref=" atom "/>
      <xs:element ref=" molecule "/>
      <xs:element ref=" and "/>
      <xs:element ref=" or "/>
      <xs:element ref=" neg "/>
      <xs:element ref=" naf "/>
      <xs:element ref=" implies "/>
      <xs:element ref=" impliedBy "/>
      <xs:element ref=" equivalent "/>
      <xs:element ref=" forall "/>
      <xs:element ref=" exists "/>
   </xs:choice>
</xs:complexType>
```

top

Element: and

Name	and
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Conjunction

XML Instance Representation

```
<and>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</and>
```

Schema Component Representation

```
<xs:element name="and" type=" binaryComplexExpressionType "/>
```

top

Element: or

Name	or
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Disjunction

XML Instance Representation

```
<or>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</or>
```

Schema Component Representation

```
<xs:element name="or" type=" binaryComplexExpressionType "/>
```

top

Element: neg

Name	neg
Type	complexExpressionType
Nillable	no
Abstract	no
Documentation	Classical negation

XML Instance Representation

```
<neg>
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</neg>
```

Schema Component Representation

```
<xs:element name="neg" type=" complexExpressionType "/>
```

top

Element: naf

Name	naf
Type	complexExpressionType
Nillable	no
Abstract	no
Documentation	Negation-as-failure

XML Instance Representation

```
<naf>
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</naf>
```

Schema Component Representation

```
<xs:element name="naf" type=" complexExpressionType "/>
```

top

Element: implies

Name	implies
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Right implication (corresponds with necessary conditions). The first formula nested inside the 'implies' element, implies the second formula. For an implication, exactly two formulas must be nested inside the implication element.

XML Instance Representation

```
<implies>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</implies>
```

Schema Component Representation

```
<xs:element name="impliedBy" type=" binaryComplexExpressionType "/>
```

top

Element: impliedBy

Name	impliedBy
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Left implication (corresponds with sufficient conditions). The first formula nested inside the 'implies' element, is implied by the second formula. For an implication, exactly two formulas must be nested inside the implication element.

XML Instance Representation

```
<impliedBy>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</impliedBy>
```

Schema Component Representation

```
<xs:element name="impliedBy" type=" binaryComplexExpressionType "/>
```

top

Element: equivalent

Name	equivalent
Type	binaryComplexExpressionType
Nillable	no
Abstract	no
Documentation	Dual implication (corresponds with necessary and sufficient conditions). The first formula nested inside the 'implies' element, is implied by the second formula. For an implication, exactly two formulas must be nested inside the implication element.

XML Instance Representation

```
<equivalent>
```

```
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</equivalent>
```

Schema Component Representation

```
<xs:element name="equivalent" type=" binaryComplexExpressionType "/>
```

top

Element: forall

Name	forall
Type	quantifiedComplexExpressionType
Nillable	no
Abstract	no
Documentation	Universal quantification.

XML Instance Representation

```
<forall>
   <var> wsmlVariable </var> [1..*]
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</forall>
```

Schema Component Representation

```
<xs:element name="forall" type=" quantifiedComplexExpressionType "/>
```

top

Element: exists

Name	exists
Type	quantifiedComplexExpressionType

Nillable	no
Abstract	no
Documentation	Existential quantification

XML Instance Representation

```
<exists>
   <var> wsmlVariable </var> [1..*]
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</exists>
```

Schema Component Representation

```
<xs:element name="exists" type=" quantifiedComplexExpressionType "/>
```

top

Complex Type: quantifiedComplexExpressionType

Super-types:	None
Sub-types:	None

Name	quantifiedComplexExpressionType
Abstract	no
Documentation	For now the complexExpressionType is repeated after the 'var' element, because I don't know how to include it *after* 'var'.

XML Instance Representation

```
<...>
   <var> wsmlVariable </var> [1..*]
   Start Choice [1]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="quantifiedComplexExpressionType">
  <xs:sequence>
    <xs:element name="var" type=" wsmlVariable "
    maxOccurs="unbounded"/>
    <xs:choice>
      <xs:element ref=" atom "/>
      <xs:element ref=" molecule "/>
      <xs:element ref=" and "/>
      <xs:element ref=" or "/>
      <xs:element ref=" neg "/>
      <xs:element ref=" naf "/>
      <xs:element ref=" implies "/>
      <xs:element ref=" impliedBy "/>
      <xs:element ref=" equivalent "/>
      <xs:element ref=" forall "/>
      <xs:element ref=" exists "/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>
```

top

Complex Type: complexExpressionType

Super-types:	None
Sub-types:	None

Name	complexExpressionType
Abstract	no
Documentation	The basic type for WSML-Full expressions, allowing exactly one expression

XML Instance Representation

```
<...>
  Start Choice [1]
    <atom> ... </atom> [1]
    <molecule> ... </molecule> [1]
    <and> ... </and> [1]
    <or> ... </or> [1]
    <neg> ... </neg> [1]
    <naf> ... </naf> [1]
    <implies> ... </implies> [1]
    <impliedBy> ... </impliedBy> [1]
    <equivalent> ... </equivalent> [1]
    <forall> ... </forall> [1]
    <exists> ... </exists> [1]
  End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="complexExpressionType">
  <xs:choice>
    <xs:element ref=" atom "/>
    <xs:element ref=" molecule "/>
    <xs:element ref=" and "/>
    <xs:element ref=" or "/>
    <xs:element ref=" neg "/>
```

```
      <xs:element ref=" naf "/>
      <xs:element ref=" implies "/>
      <xs:element ref=" impliedBy "/>
      <xs:element ref=" equivalent "/>
      <xs:element ref=" forall "/>
      <xs:element ref=" exists "/>
   </xs:choice>
</xs:complexType>
```

top

Complex Type: binaryComplexExpressionType

Super-types:	None
Sub-types:	None

Name	binaryComplexExpressionType
Abstract	no
Documentation	The basic type for WSML-Full expressions, allowing exactly two expressions to each side of the operator

XML Instance Representation

```
<...>
   Start Choice [2..2]
      <atom> ... </atom> [1]
      <molecule> ... </molecule> [1]
      <and> ... </and> [1]
      <or> ... </or> [1]
      <neg> ... </neg> [1]
      <naf> ... </naf> [1]
      <implies> ... </implies> [1]
      <impliedBy> ... </impliedBy> [1]
      <equivalent> ... </equivalent> [1]
      <forall> ... </forall> [1]
      <exists> ... </exists> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="binaryComplexExpressionType">
   <xs:choice minOccurs="2" maxOccurs="2">
      <xs:element ref=" atom "/>
      <xs:element ref=" molecule "/>
      <xs:element ref=" and "/>
      <xs:element ref=" or "/>
      <xs:element ref=" neg "/>
      <xs:element ref=" naf "/>
      <xs:element ref=" implies "/>
      <xs:element ref=" impliedBy "/>
      <xs:element ref=" equivalent "/>
      <xs:element ref=" forall "/>
      <xs:element ref=" exists "/>
   </xs:choice>
</xs:complexType>
```

top

Element: atom

Name	atom
Type	Locally-defined complex type
Nillable	no
Abstract	no
Documentation	An atom denotes a relation with an n-ary domain, where n is the arity of the predicate

XML Instance Representation

```
<atom
 name="anySimpleType [1]">
   <arg> termType </arg> [0..*]
</atom>
```

Schema Component Representation

```
<xs:element name="atom">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="arg" type=" termType " minOccurs="0"
         maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" use="required"/>
   </xs:complexType>
</xs:element>
```

top

Complex Type: termType

Super-types:	None
Sub-types:	None

Name	termType
Abstract	no
Documentation	A term can be a IRI or a variable, or a constructed term (corresponsing to a function symbol), where the arguments are terms themselves. In the latter case, the name may not be a variable.

XML Instance Representation

```
<...
 name=" xs:string [1]">
   <arg> termType </arg> [0..*]
</...>
```

Schema Component Representation

```
<xs:complexType name="termType">
   <xs:sequence>
      <xs:element name="arg" type=" termType " minOccurs="0"
      maxOccurs="unbounded"/>
   </xs:sequence>
```

```
   <xs:attribute name="name" type=" xs:string " use="required"/>
</xs:complexType>
```

top

Element: molecule

Name	molecule
Type	Locally-defined complex type
Nillable	no
Abstract	no

XML Instance Representation

```
<molecule>
   <term> termType </term> [1]
   <attributeDefinition
    type=" xs:string (value comes from list:
   {'inferring'|'constraining'}) [0..1]"> [0..*]
      <name> termType </name> [1]
      <type> termType </type> [1..*]
   </attributeDefinition>
   <attributeValue> [0..*]
      <name> termType </name> [1]
      <value> termType </value> [1..*]
   </attributeValue>
   <isa
    type=" xs:string (value comes from list:
   {'memberOf'|'subConceptOf'}) [0..1]"> [0..1]
      <term> termType </term> [1..*]
   </isa>
</molecule>
```

Schema Component Representation

```
<xs:element name="molecule">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="term" type=" termType "/>
         <xs:element name="attributeDefinition" minOccurs="0"
         maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="name" type=" termType "/>
                  <xs:element name="type" type=" termType "
                  maxOccurs="unbounded"/>
               </xs:sequence>
               <xs:attribute name="type" default="constraining">
                  <xs:simpleType>
                     <xs:restriction base=" xs:string ">
                        <xs:enumeration value="inferring"/>
                        <xs:enumeration value="constraining"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:attribute>
            </xs:complexType>
         </xs:element>
         <xs:element name="attributeValue" minOccurs="0"
         maxOccurs="unbounded">
            <xs:complexType>
               <xs:sequence>
                  <xs:element name="name" type=" termType "/>
```

```
            <xs:element name="value" type=" termType "
            maxOccurs="unbounded"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="isa" minOccurs="0">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="term" type=" termType "
            maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:attribute name="type">
            <xs:simpleType>
              <xs:restriction base=" xs:string ">
                <xs:enumeration value="memberOf"/>
                <xs:enumeration value="subConceptOf"/>
              </xs:restriction>
            </xs:simpleType>
          </xs:attribute>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

top

Generated by xs3p.