

Content Models for RuleML

**Tara Athan, Harold Boley, Tshering Dema, David Hirtle,
Omair Shafiq, Derek Smith**

2011-10-19, version [1.0](#)

Introduction

This document is a collection of content models, i. e. the content permitted within a particular element, for all RuleML elements and attributes as of version 1.0 as defined by the XSD schemas, organized alphabetically by module name. Each module is a grouping of related (XML) elements and/or attributes (prefixed with “@”). Under each element name, the attributes allowed are listed in the first line, with suffix ? to indicate optional attributes. We are able to do this because the content models of all attributes are independent of context. Following the attribute list, the element and text portion of the content model is given in EBNF-like compact Relax NG syntax. In attributes, default values, if any, are first in the list of possible values, and are shown in bold for emphasis. See <http://www.ruleml.org/1.0/xsd/> for the actual XSD schemas and the RuleML [glossary](#) for the meaning of each tag.

The content models presented here apply only to the XSD schemas, not the Relax NG schemas of RuleML Version 1.0. For content models of the Relax NG schemas for the original fifteen named sublanguages, see <http://ruleml.org/1.0/simplified> .

Since RuleML is a family of sublanguages, it is important to note that the content model of a given node often varies according to the current sublanguage. In such cases, all variations of the content model are provided along with the corresponding sublanguage(s). The modularization of RuleML, including all sublanguages, is explained at <http://www.ruleml.org/modularization>.

Content models may also vary depending on context, i.e. surrounding elements (especially parent elements). In these cases, the content models are listed under a heading such as “within x...” where x indicates the context.

Any element in a sublanguage may be used within Reify or as the starting element of the grammar (the document root, or the starting element of a section in the RuleML namespace within a multi-namespace document) except as stated otherwise. As implemented in the XSD schemas, when elements have context-dependent content models, at most one of the patterns is allowed in Reify or start - otherwise the content model will be non-deterministic, which is not allowed in XSD 1.0.

In a few cases, there is a discrepancy between the content model as implemented in the XSD schema and the intended sublanguage, principally due to limitations of the XSD language. Discrepancies are denoted by color-coded highlighting:

- yellow indicates parts of the content model that are implemented in the XSD schemas but not consistent with the intended sublanguage;
- blue indicates parts of the content model of the intended sublanguage that are not implemented in the XSD schemas;
- pink indicates parts of the content model that are associated with languages outside of Deliberation RuleML.

The content model as implemented in XSD may be obtained by including yellow highlighted sections and deleting blue and pink.

The content model of the intended sublanguage(s) may be obtained by including blue highlighted sections and deleting yellow and pink.

For clarification on any RuleML-related topic, including this document, the RuleML-all [mailing list](#) may be quite helpful. The RuleML [tutorial](#) serves as an introduction.

Index

<u>Introduction</u>	2
<u>Index</u>	3
<u>atom</u>	6
Atom	6
degree	6
op (context sensitive; see also the holog and expr modules)	6
within Atom, Reify and start	6
Rel	6
<u>connective</u>	7
if (context sensitive)	7
within Implies, Reify and start	7
within Entails	8
then (context sensitive)	8
within Implies, Reify and start	8
within Entails	9
Implies	9
Entails	9
Equivalent	9
torso	9
Rulebase	10
And	10
Or	10
formula (context sensitive; see also performative and quantifier modules)	10
within Rulebase	10
within And/Or	11
@mapMaterial	12
@material	12
@mapDirection	12
@direction	12
@mapClosure	12
@closure	12
<u>desc</u>	13
oid	13
<u>equality</u>	14
Equal	14
left	14
right	14
@oriented	14
@val	14
<u>expr</u>	15
Expr	15
op (context sensitive; see also the atom and holog modules)	15
within Expr	15
Fun	15
Plex (context sensitive)	15
within Atom, Plex, slot, Reify and start	15
within repo	15

within resl.....	15
@per.....	15
frame	16
Set.....	16
InstanceOf.....	16
SubclassOf.....	16
Signature.....	16
Get.....	16
SlotProd.....	16
holog	17
Uniterm.....	17
op (context sensitive; see also the atom and expr modules).....	17
within Uniterm and Signature.....	17
Const.....	17
@minCard.....	17
@maxCard.....	17
iri	18
@iri.....	18
naf	19
Naf.....	19
weak.....	19
neg	20
Neg.....	20
strong.....	20
performative	21
RuleML.....	21
act.....	21
Assert.....	21
Retract.....	21
Query.....	21
formula (context sensitive; see also connective and quantifier modules).....	21
within Assert and Retract... ..	21
within Query.....	22
quantifier	24
Forall.....	24
Exists.....	24
declare.....	24
formula (context sensitive; see also the connective and performative modules).....	24
within Forall... ..	24
within Exists... ..	25
rest	26
repo.....	26
resl.....	26
slot	27
slot (context sensitive).....	27
within Atom, Expr, Plex, Uniterm, Reify and start.....	27
within Signature, Atom-frame.....	27
@card.....	27
@weight.....	27
term	28

arg	28
Ind	28
Data	28
Var	28
Skolem	29
Reify	29
@type	29
@index	29

atom

Atom

attributes: @closure?

in bindatagroundlog, bindatagroundfact and bindatalog:

```
( oid?, degree?, (op | Rel), slot*, ((arg | arg_content), (arg | arg_content), slot*)? )
```

in datalog, nafdatalog, nafnegdatalog, and negdatalog:

```
( oid?, degree?, (op | Rel), slot*, ((arg | arg_content)+, slot*)? )
```

in hornlog & up (except framehohornlogeq):

```
( oid?, degree?, (op | Rel), slot*, (((arg | arg_content)+, repo?) | repo), slot*)?, res1? )
```

in framehohornlogeq (except in Reify or start):

```
( oid, ( op | op_content )?, slot* )
```

within Reify and start in SWSL languages :

```
oid?, degree?, (op | Rel), slot* (((arg | Var | Skolem | Reify | Const | Uniterm)+, repo?) |  
repo), slot*)?, res1?
```

degree

attributes: none

in all sublanguages:

```
(Data)
```

op (context sensitive; see also the holog and expr modules)

attributes: none

within Atom, Reify and start...

in all sublanguages (except SWSL languages):

```
( Rel )
```

Rel

attributes: @iri?

in all sublanguages:

```
( text )
```

connective

if (context sensitive)

attributes: none

within Implies, Reify and start...

in all sublanguages (except bindatagroundfact):

```
( if_implies_content )
```

where if_implies_content =

in datalog & down and hornlog, dishornlog:

```
( Atom | And | Or )
```

in negdatalog:

```
( Atom | And | Or | Neg )
```

in nafdatalog & nafhornlog:

```
( Atom | And | Or | Naf )
```

in nafnegdatalog:

```
( Atom | And | Or | Neg | Naf )
```

in hornlog_{eq}:

```
( Atom | And | Or | Equal )
```

in hohornlog:

```
( And | Or | Naf | Uniterm | Neg )
```

in hohornlog_{eq}:

```
( And | Or | Naf | Uniterm | Neg | Equal )
```

in framehohornlog_{eq}:

```
( And | Or | Naf | Uniterm | Neg | Equal | Atom | InstanceOf | SubclassOf |  
Signature )
```

in folog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffolog:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists )
```

in folog_{eq}:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

in naffolog_{eq}:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists | Equal )
```


within Entails ...

in all sublanguages:

(if_entails_content)

where if_entails_content =

in all sublanguages:

(Rulebase)

then (context sensitive)

attributes: none

within Implies, Reify and start...

in all sublanguages (except bindatagroundfact):

(then_implies_content)

where then_implies_content =

in datalog & down and hornlog:

(Atom)

in negdatalog:

(Atom | Neg)

in nafdatalog & nafhornlog:

(Atom | Naf)

in nafnegdatalog:

(Atom | Neg | Naf)

in hornlog_{eq}:

(Atom | Equal)

in hohornlog:

(Naf | Uniterm | Neg)

in hohornlog_{eq}:

(Naf | Uniterm | Neg | Equal)

in framehohornlog_{eq}:

(Naf | Uniterm | Neg | Equal | Atom | InstanceOf | SubclassOf | Signature)

in dishornlog:

(Atom | And | Or)

in folog:

(Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists)

in naffolog:

(Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists)

in fologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

in naffologeq:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists | Equal )
```

within Entails...

in all sublanguages:

```
( then_entails_content )
```

where then_entails_content =

in all sublanguages:

```
( Rulebase )
```

Implies

attributes: @closure?, @direction?, @material? (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, ((then, if) | (if, then) | (if_implies_content, then_implies_content)) )
```

Entails

attributes: none

in all sublanguages:

```
( oid?, (if | Rulebase), (then | Rulebase) )
```

Equivalent

attributes: @closure? (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, ((torso, torso) | ( torso_content, torso_content )) )
```

torso

attributes: none

in all sublanguages:

```
( torso_content )
```

where torso_content =

in datalog & down and up to (and including) dishornlog:

```
( Atom )
```

in dishornlog:

```
( Atom | Or )
```

in hornlog_{eq}:

```
( Atom | Equal )
```

in hohornlog:

```
( Uniterm )
```

in hohornlog_{eq}:

```
( Uniterm | Equal )
```

in framehohornlog_{eq}:

```
( Atom | Uniterm | InstanceOf | SubclassOf | Signature | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in folog_{eq} & naffolog_{eq}:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

Rulebase

attributes: @mapDirection?, @mapClosure? and @mapMaterial?

in all sublanguages:

```
( oid?, (formula | formula_rulebase_content)* )
```

And

attributes {within Query only: @closure?} (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_andor_content)* )
```

Or

attributes {within Query only: @closure?} (+ @mapDirection?, @mapClosure? and @mapMaterial? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_andor_content)* )
```

formula (context sensitive; see also performative and quantifier modules)

attributes: none

within Rulebase...

```
( formula_rulebase_content )
```

where formula_rulebase_content =

in bindatagroundfact:

(Atom)

in bindatagroundlog:

(Atom | Implies | Equivalent)

in bindatalog, datalog, nafdatalog, hornlog, nafhornlog dishornlog:

(Atom | Implies | Equivalent | Forall)

in negdatalog, nafnegdatalog:

(Atom | Implies | Equivalent | Forall | Neg)

in hornlog:

(Atom | Implies | Equivalent | Forall | Equal)

in hohornlog:

(Implies | Equivalent | Forall | Uniterm | Neg)

in hohornlog:

(Implies | Equivalent | Forall | Uniterm | Neg | Equal)

in framehohornlog:

(Implies | Equivalent | Forall | Uniterm | Neg | Equal | Atom | InstanceOf | SubclassOf |
Signature)

in folog, naffolog:

(Atom | Implies | Equivalent | Forall | And | Or | Neg | Exists)

in fologeq, naffologeq:

(Atom | Implies | Equivalent | Forall | And | Or | Neg | Exists | Equal)

within And/Or...

(formula_andor_content)

where formula_andor_content =

in datalog & down, hornlog and dishornlog:

(Atom | And | Or)

in negdatalog:

(Atom | And | Or | Neg)

in nafdatalog and nafhornlog:

(Atom | And | Or | Naf)

in nafnegdatalog:

(Atom | And | Or | Naf | Neg)

in hornlog:

(Atom | And | Or | Equal)

in hohornlog:

```
( And | Or | Naf | Uniterm | Neg )
```

in hohornlog_{eq}:

```
( And | Or | Naf | Uniterm | Neg | Equal )
```

in framehohornlog_{eq}:

```
( And | Or | Naf | Uniterm | Neg | Equal | Atom | InstanceOf | SubclassOf |  
Signature )
```

in folog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffolog:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists )
```

in folog_{eq}:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

in naffolog_{eq}:

```
( Atom | And | Or | Neg | Naf | Implies | Equivalent | Forall | Exists | Equal )
```

@mapMaterial

```
( yes | no )
```

@material

```
( yes | no )
```

@mapDirection

```
( bidirectional | forward | backward )
```

@direction

```
( bidirectional | forward | backward )
```

@mapClosure

```
( universal | existential )
```

@closure

```
( universal | existential )
```

desc

oid

attributes: none

in bindatagroundfact and bindatagroundlog :

```
( arg_content | Var )
```

in all other sublanguages except framehohornlogeq:

```
( arg_content )
```

framehohornlogeq:

```
( op_content )
```

equality

Equal

attributes: none

in hornlog_{eq}, folo_{eq}, naffolo_{eq}, hohornlog_{eq}, framehohornlog_{eq}
(oid?, degree?, ((left, right) | (arg_content, arg_content)))

left

attributes: none

in hornlog_{eq}, folo_{eq}, naffolo_{eq}, hohornlog_{eq}, framehohornlog_{eq}:
(arg_content)

right

same as left

@oriented

(no | yes)

@val

(0.. | 1)

expr

Expr

attributes: @type?

in hornlog & up (except hohornlog, etc):

```
( oid?, (op | Fun), slot*, (((arg | arg_content)+, repo?) | repo), slot*)?, resl? )
```

op (context sensitive; see also the atom and holog modules)

within Expr:

attributes: none

in hornlog & up (except SWSL languages):

```
( Fun )
```

Fun

attributes: @iri?

in hornlog & up (except SWSL languages):

```
( text )
```

Plex (context sensitive)

attributes: none

within Atom, Plex, slot, Reify and start...

in hornlog & up :

```
( oid?, slot*, (((arg | arg_content)+, repo?, slot*, resl?)? | (repo, slot*, resl?) | resl) )
```

within repo...

in hornlog & up:

```
( ( arg | arg_content)* , repo? )
```

within resl...

in hornlog & up including hohornlog, etc. :

```
( slot*, resl? )
```

@per

```
( copy | open | value | effect | model )
```


frame

Set

attributes: none
in framehohornlogeq:
`(arg_content*)`

InstanceOf

attributes: none
in framehohornlogeq:
`(arg_content, arg_content)`

SubclassOf

attributes: none
in framehohornlogeq:
`(arg_content, arg_content)`

Signature

attributes: none
in framehohornlogeq:
`(oid, (op | op_content)?, slot*)`

Get

attributes: none
in framehohornlogeq:
`(oid, SlotProd)`

SlotProd

attributes: none
in framehohornlogeq:
`((arg_content)+)`

holog

Uniterm

attributes: none

in hohornlog, hohornlogeq & framehohornlogeq:

```
( oid?, (op | op_content), slot*, res1?, (((arg | arg_content)+, repo?) | repo), slot*, res1?))  
( oid?, (op | op_content), slot*, (((arg | arg_content)+, repo?) | repo), slot*)?, res1?)
```

op (context sensitive; see also the atom and expr modules)

attributes: none

within Uniterm and Signature...

in hohornlog & up:

```
( op_content )
```

where op_content =

```
( Const | Skolem | Var | Reify | Uniterm )
```

Const

attributes: @iri?, @type?

in hohornlog & up:

```
( text )
```

@minCard

in hohornlog & up:

```
( xsd:nonNegativeInteger )
```

@maxCard

in hohornlog & up:

```
( xsd:nonNegativeInteger )
```

iri

@iri

in all sublanguages:

(`xsd:anyURI`)

naf

Naf

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, ( weak | weak_content) )
```

weak

attributes: none

in all sublanguages:

```
( weak_content )
```

where weak_content =

in nafdatalog:

```
( Atom )
```

in nafnegdatalog:

```
( Atom | Neg )
```

in hohornlog

```
( Uniterm )
```

in naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in naffologeq:

```
( Atom | And| Or| Neg| Implies| Equivalent| Forall| Exists| Equal )
```

neg

Neg

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (strong | strong_content) )
```

strong

attributes: none

in all sublanguages:

```
( strong_content )
```

where strong_content =

in negdatalog and nafnegdatalog:

```
( Atom )
```

in hohornlog:

```
( Uniterm )
```

in hohornlogeq & up:

```
( Uniterm | Equal )
```

in folog and naffolog:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists )
```

in fologeq and naffologeq:

```
( Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal )
```

performative

RuleML

attributes: none

in all sublanguages:

```
( oid?, (act | Assert | Query | Retract)* )
```

act

attributes: index

in all sublanguages:

```
( Assert | Query | Retract )
```

Assert

attributes: @mapDirection?, @mapClosure? and @mapMaterial?

in all sublanguages:

```
( oid?, (formula | formula_assert_retract)* )
```

Retract

same as Assert

Query

attributes: @closure? (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages:

```
( oid?, (formula | formula_query)* )
```

formula (context sensitive: see also connective and quantifier modules)

attributes: none

within Assert and Retract...

```
( formula_assert_retract )
```

where formula_assert_retract =

in bindatagroundfact:

```
( Rulebase | Atom | Entails )
```

in bindatagroundlog:

(Rulebase | Atom | Implies | Equivalent | Entails)

in bindatalog, datalog, hornlog & dishornlog, nafdatalog, nafhornlog:

(Rulebase | Atom | Implies | Equivalent | Entails | Forall)

in negdatalog and nafnegdatalog:

(Rulebase | Atom | Implies | Equivalent | Entails | Forall | Neg)

in hornlog_{eq}:

(Rulebase | Atom | Implies | Equivalent | Entails | Forall | Equal)

in hohornlog:

(Rulebase | Implies | Equivalent | Entails | Forall | Uniterm | Neg)

in hohornlog_{eq}:

(Rulebase | Implies | Equivalent | Entails | Forall | Uniterm | Neg | Equal)

in framelihornlog_{eq}:

(Rulebase | Implies | Equivalent | Entails | Forall | Uniterm | Neg | Equal | Atom
| InstanceOf | SubclassOf | Signature)

in folog and naffolog:

(Rulebase | Atom | Implies | Equivalent | Entails | Forall | And | Or | Neg | Exists
)

in folog_{eq} and naffolog_{eq}:

(Rulebase | Atom | Implies | Equivalent | Entails | Forall | And | Or | Neg |
Exists | Equal)

within Query...

(formula_query)

where formula_query =

in bindatagroundfact and bindatagroundlog:

(Rulebase | And | Or | Atom | Entails_)

in bindatalog, datalog, hornlog, dishornlog:

(Rulebase | Atom | And | Or | Entails | Exists)

in nafdatalog, nafhornlog:

(Rulebase | Atom | And | Or | Entails | Exists | Naf)

in negdatalog:

(Rulebase | Atom | And | Or | Entails | Exists | Neg)

in nafnegdatalog:

(Rulebase | Atom | And | Or | Entails | Exists | Neg | Naf)

in framelihornlog_{eq}:

(Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg | Equal | Atom | InstanceOf |
SubclassOf | Signature)

in hohornlog:

```
( Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg )
```

in hohornlogeq:

```
( Rulebase | And | Or | Entails | Exists | Naf | Uniterm | Neg | Equal )
```

in folog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall )
```

in fologeq:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Equal )
```

in naffolog:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Naf )
```

in naffologe:

```
( Rulebase | Atom | And | Or | Entails | Exists | Neg | Implies | Equivalent | Forall | Naf |  
  Equal )
```


quantifier

Forall

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages (except bindatagroundfact and bindatagroundlog):

```
( oid?, (declare | Var)+, (formula | formula_forall) )
```

Exists

attributes: none (+ @mapDirection?, @mapMaterial? and @mapClosure? in folog & up)

in all sublanguages (except bindatagroundfact and bindatagroundlog):

```
( oid?, (declare | Var)+, (formula | formula_exists) )
```

declare

attributes: none

in all sublanguages (except bindatagroundfact and bindatagroundlog):

```
( Var )
```

formula (context sensitive; see also the connective and performative modules)

attributes: none

in all sublanguages (except bindatagroundfact and bindatagroundlog):

within Forall...

```
( formula_forall )
```

where formula_forall =

in bindatalog, datalog & up to (including) hornlog and dishornlog:

```
( Atom | Implies | Equivalent | Forall )
```

in hornlog:

```
( Atom | Implies | Equivalent | Forall | Equal )
```

in hohornlog:

```
( Uniterm | Implies | Equivalent | Forall )
```

in hohornlog:

```
( Uniterm | Implies | Equivalent | Forall | Equal )
```

in framehohornlog:

```
( Atom | Uniterm | InstanceOf | SubclassOf | Signature | Implies | Equivalent | Forall | Equal )
```

in folog and naffolog:

(Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists)

in fologeq and naffologeq:

(Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal)

within Exists...

(formula_exists)

where formula_exists =

in bindatalog, datalog & up to (including) hornlog and dishornlog:

(Atom | And | Or | Exists)

in hornlogeq:

(Atom | And | Or | Exists | Equal)

in hohornlog:

(Uniterm | And | Or | Exists)

in hohornlogeq:

(Uniterm | And | Or | Exists | Equal)

in framehohornlogeq:

(Atom | Uniterm | InstanceOf | SubclassOf | Signature | And | Or | Exists | Equal)

in folog and naffolog:

(Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists)

in fologeq and naffologeq:

(Atom | And | Or | Neg | Implies | Equivalent | Forall | Exists | Equal)

rest

repo

attributes: none

in hornlog & up:

(Var | Plex)

resl

attributes: none

in hornlog & up:

(Var | Plex)

slot

slot (context sensitive)

attributes: @card?, @weight?

within Atom, Expr, Plex, Uniterm, Reify and start

in bindatagroundlog, & up (except for the SWSL languages):

```
( (Ind | Data), arg_content )
```

in hohornlog & hohornlogeq:

```
( ( Const | Uniterm ), arg_content )
```

in framehohornlogeq:

```
( ( Const | Uniterm | Get ), arg_content )
```

within Signature, Atom-frame...

attributes: (+ @minCard? and @maxCard?)

in framehohornlogeq:

```
( ( Const | Uniterm | Get ), arg_content? )
```

@card

in all sublanguages:

```
( xsd:nonNegativeInteger )
```

@weight

in all sublanguages:

```
( xsd:decimal { minInclusive = "0" maxInclusive = "1" } )
```

term

arg

attributes: @index

in all sublanguages:

(arg_content)

where arg_content =

in bindatalog, datalog & up to hornlog:

(Ind | Data | Skolem | Var | Reify)

in bindatagroundlog and bindatagroundfact:

(Ind | Data | Skolem | Reify)

in hornlog & up (except hohornlog, etc):

(Ind | Data | Skolem | Var | Reify | Expr | Plex)

in hohornlog & hohornlogeq:

(Const | Skolem | Var | Reify | Uniterm)

in framehohornlogeq:

(Const | Skolem | Var | Reify | Uniterm | Get | Set)

Ind

attributes: @iri?, @type?

in all sublanguages:

(text)

Data

attributes: @xsi:type?

in all sublanguages:

(xsd:anyType) optionally restricted to the datatype specified as the value of @xsi:type

Var

attributes: @type?

in all sublanguages (except bindatagroundfact and bindatagroundlog):

(text)

Skolem

attributes: *@type*?

in all sublanguages:

(text)

Reify

attributes: none

in all sublanguages:

(xsd:anyComplexType?) restricted to globally-defined elements in the RuleML namespace with strict validation of content

@type

in all sublanguages:

(text)

@index

in all sublanguages:

(xsd:positiveInteger)