Atelier B - POG format documentation

Schema Document Properties

Properties

Target Namespace	https://www.atelierb.eu/Formats/pog
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 namespace. • By default, local attribute declarations have no namespace.

Documentation

The purpose of this document is to describe and illustrate the POG format: an XML representation of the proof obligations for a B component or for an Event-B component.

- This documentation corresponds to version 1.0 of the format.
- The root element is always a Proof_Obligations.

Proof obligations are essentially combinations of B predicates. An important part of the format describes how B predicates, expressions and types are represented. This description references the B Language Reference Manual, version 1.8.10 as *BLRM*.

Licensing

Declared Namespaces

Prefix	**Namespace**
Default namespace	https://www.atelierb.eu/Formats/pog
xml	http://www.w3.org/XML/1998/namespace
xs	http://www.w3.org/2001/XMLSchema

Schema Component Representation

```
<xs:schema targetNamespace="https://www.atelierb.eu/Formats/pog" elementFormDefault="qualified">
...
</xs:schema>
```

Global Declarations

Element: Binary_Exp

Properties

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

Documentation

Represents a binary expression.

- Two child elements represent the arguments.
- Attribute `op` represents the operator and shall be a binary_exp_op.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Binary_Exp
 op="binary_exp_op" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   Start Group: exp_group [2..2]
      Start Choice [1]
         <Unary_Exp> ... </Unary_Exp> [1]
         <Binary_Exp> ... </Binary_Exp> [1]
         <Ternary_Exp> ... </Ternary_Exp> [1]
         <Nary_Exp> ... </Nary_Exp> [1]
         <Boolean_Literal> ... </Boolean_Literal> [1]
         <Boolean_Exp> ... </Boolean_Exp> [1]
         <EmptySet> ... </EmptySet> [1]
         <EmptySeq> ... </EmptySeq> [1]
         <Id> ... </Id> [1]
         <Integer_Literal> ... </Integer_Literal> [1]
         <Quantified_Exp> ... </Quantified_Exp> [1]
         <Quantified_Set> ... </Quantified_Set> [1]
         <STRING_Literal> ... </STRING_Literal> [1]
         <Struct> ... </Struct> [1]
         <Record> ... </Record> [1]
         <Real_Literal> ... </Real_Literal> [1]
         <Record_Update> ... </Record_Update> [1]
         <Record_Field_Access> ... </Record_Field_Access> [1]
      End Choice
   End Group: exp_group
</Binary_Exp>
```

Schema Component Representation

```
<xs:element name="Binary_Exp">
   <xs:complexType>
      <xs:group ref="exp_group" minOccurs="2" maxOccurs="2"/>
      <xs:attribute name="op" type="binary_exp_op" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Binary_Pred

Properties

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

Documentation

Represents a binary predicate.

- Two child elements represent the arguments.
- Attribute op represents the operator and shall be a binary_pred_op.

XML Instance Representation

```
<Binary_Pred
 op="binary_pred_op" [1]
>
   Start Group: pred_group [2..2]
      Start Choice [1]
         <Binary_Pred> ... </Binary_Pred> [1]
         <Exp_Comparison> ... </Exp_Comparison> [1]
         <Quantified_Pred> ... </Quantified_Pred> [1]
         <Unary_Pred> ... </Unary_Pred> [1]
         <Nary_Pred> ... </Nary_Pred> [1]
      End Choice
   End Group: pred_group
</Binary_Pred>
```

Schema Component Representation

```
<xs:element name="Binary_Pred">
   <xs:complexType>
      <xs:group ref="pred_group" minOccurs="2" maxOccurs="2"/>
      <xs:attribute name="op" type="binary_pred_op" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Boolean_Exp

Properties

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

Documentation

Representation the conversion of a predicate to a Boolean expression.

- A single child element represents the converted predicate.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Boolean_Exp
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   Start Choice [1]
      <Binary_Pred> ... </Binary_Pred> [1]
      <Exp_Comparison> ... </Exp_Comparison> [1]
      <Quantified_Pred> ... </Quantified_Pred> [1]
      <Unary_Pred> ... </Unary_Pred> [1]
      <Nary_Pred> ... </Nary_Pred> [1]
   End Choice
</Boolean_Exp>
```

Schema Component Representation

```
<xs:element name="Boolean_Exp">
   <xs:complexType>
      <xs:group ref="pred_group"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Boolean_Literal

Properties

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

Documentation

Represents a Boolean literal expression.

- Attribute `value` is the represented literal and is of type boolean_literal_type.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Boolean_Literal
 value="boolean_literal_type" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="Boolean_Literal">
   <xs:complexType>
      <xs:attribute name="value" type="boolean_literal_type" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Define

Properties

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

Documentation

This element represents part of the context. The name attribute identifies which part of the context the current element represents. The possible values for this attribute are :

- `"B definitions"` : Definitions of predefined sets `NAT` and `INT`.
- `"ctx"` : SETS and PROPERTIES from seen components and their included components;
- `"seext"` : INVARIANT and ASSERTIONS from seen components and their included components;
- `"inv"` : INVARIANT of the component;
- `"ass"` : ASSERTIONS of the current component;
- `"lprp"` : SETS and PROPERTIES of the current component;
- `"inprp"` : SETS and PROPERTIES of the included components of the current component;
- `"inext"` : INVARIANT and ASSERTIONS clauses of the included components of the current component;
- `"cst"` : CONSTRAINTS clause of the current component;
- `"sets"` : SETS clause of the current component;

If the current component is a refinement or an implementation:

- `"mchcst"` : CONSTRAINTS clause of the component refined by the current component;
- `"aprp"` : SETS and PROPERTIES clauses of all the refined components as well as their included components;
- `"abs"` : INVARIANT and ASSERTIONS clauses of all the refined components as well as their included components;

If the current component is an implementation:

- `"imlprp"` : SETS, PROPERTIES and VALUES clauses of the current component;
- `"imprp"` : SETS and PROPERTIES clauses of all the imported components as well as their included components;
- `"imext"` : INVARIANT and ASSERTIONS clauses of all the imported components as well as their included components.

XML Instance Representation

```
<Define
 name="xs:string" [1]
 hash="xs:nonNegativeInteger" [1]
>
   <Set> ... </Set> [0..*]
   Start Group: pred group [0..*]
      Start Choice [1]
         <Binary Pred> ... </Binary Pred> [1]
         <Exp Comparison> ... </Exp Comparison> [1]
         <Quantified Pred> ... </Quantified Pred> [1]
         <Unary Pred> ... </Unary Pred> [1]
         <Nary Pred> ... </Nary Pred> [1]
      End Choice
   End Group: pred group
</Define>
```

Schema Component Representation

```
<xs:element name="Define">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Set" minOccurs="0" maxOccurs="unbounded"/>
         <xs:group ref="pred group" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="hash" type="xs:nonNegativeInteger" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Definition

Properties

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

Documentation

Represents a reference to an element Define. The attribute `name` identifies which element is referenced.

XML Instance Representation

```
<Definition
 name="xs:string" [1]
/>
```

Schema Component Representation

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

Element: EmptySeq

Properties

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

Documentation

Represents an empty sequence.

- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<EmptySeq
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="EmptySeq">
   <xs:complexType>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: EmptySet

Properties

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

Documentation

Represents an empty set.

- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<EmptySet
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="EmptySet">
   <xs:complexType>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Enumerated_Values

Properties

Name	Enumerated_Values
Type	Locally-defined complex type
Nillable	no

Abstract	no

Documentation

Represents an enumeration (see element Set).

- Child elements Id represent the enumerated identifiers.

XML Instance Representation

```
<Enumerated_Values>
   <Id> ... </Id> [1..*]
</Enumerated_Values>
```

Schema Component Representation

```
<xs:element name="Enumerated_Values">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Id" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Element: Exp_Comparison

Properties

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

Documentation

Represents a comparison expression.

- Two child elements represent the arguments to the expression.
- Attribute `op` represents the operator and is restricted to be a comparison_op.

XML Instance Representation

```
<Exp_Comparison
 op="comparison_op" [1]
>
   Start Group: exp_group [2..2]
      Start Choice [1]
         <Unary_Exp> ... </Unary_Exp> [1]
         <Binary_Exp> ... </Binary_Exp> [1]
         <Ternary_Exp> ... </Ternary_Exp> [1]
         <Nary_Exp> ... </Nary_Exp> [1]
         <Boolean_Literal> ... </Boolean_Literal> [1]
         <Boolean_Exp> ... </Boolean_Exp> [1]
         <EmptySet> ... </EmptySet> [1]
         <EmptySeq> ... </EmptySeq> [1]
         <Id> ... </Id> [1]
         <Integer_Literal> ... </Integer_Literal> [1]
         <Quantified_Exp> ... </Quantified_Exp> [1]
         <Quantified_Set> ... </Quantified_Set> [1]
         <STRING_Literal> ... </STRING_Literal> [1]
         <Struct> ... </Struct> [1]
         <Record> ... </Record> [1]
         <Real_Literal> ... </Real_Literal> [1]
         <Record_Update> ... </Record_Update> [1]
         <Record_Field_Access> ... </Record_Field_Access> [1]
      End Choice
   End Group: exp_group
</Exp_Comparison>
```

Schema Component Representation

```
<xs:element name="Exp_Comparison">
   <xs:complexType>
      <xs:group ref="exp_group" minOccurs="2" maxOccurs="2"/>
      <xs:attribute name="op" type="comparison_op" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Hypothesis

Properties

Name	Hypothesis
Type	predicate_type
Nillable	no
Abstract	no

Documentation

Represents an hypothesis common to all proof obligations in a Proof_Obligation.

XML Instance Representation

```
<Hypothesis>
   Start Choice [1]
      <Binary_Pred> ... </Binary_Pred> [1]
      <Exp_Comparison> ... </Exp_Comparison> [1]
      <Quantified_Pred> ... </Quantified_Pred> [1]
      <Unary_Pred> ... </Unary_Pred> [1]
      <Nary_Pred> ... </Nary_Pred> [1]
   End Choice
</Hypothesis>
```

Schema Component Representation

```
<xs:element name="Hypothesis" type="predicate_type"/>
```

Element: Id

Properties

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

Documentation

Represents the occurence of an identifier.

- Attribute `value` is the identifier.
- Optional attribute `suffix` is used for derived identifiers and is then a positive integer. Such identifiers are created to avoid name clashes.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Id
 value="xs:string" [1]
 suffix="xs:positiveInteger" [0..1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="Id">
   <xs:complexType>
      <xs:attribute name="value" type="xs:string" use="required"/>
      <xs:attribute name="suffix" type="xs:positiveInteger"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Integer_Literal

Properties

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

Documentation

Represents an integer literal expression.

- Attribute `value` is the represented literal and is an integer.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Integer_Literal
 value="xs:integer" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="Integer_Literal">
   <xs:complexType>
      <xs:attribute name="value" type="xs:integer" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Local_Hyp

Properties

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

Documentation

Represents an hypothesis common to some of the proof obligations in a Proof_Obligation.

Attribute `num` plays the role of identifier for referencing by individual proof obligations.

XML Instance Representation

```
<Local_Hyp
 num="xs:positiveInteger" [1]
>
   Start Choice [1]
      <Binary_Pred> ... </Binary_Pred> [1]
      <Exp_Comparison> ... </Exp_Comparison> [1]
      <Quantified_Pred> ... </Quantified_Pred> [1]
      <Unary_Pred> ... </Unary_Pred> [1]
      <Nary_Pred> ... </Nary_Pred> [1]
   End Choice
</Local_Hyp>
```

Schema Component Representation

```
<xs:element name="Local_Hyp">
   <xs:complexType>
      <xs:group ref="pred_group"/>
      <xs:attribute name="num" type="xs:positiveInteger" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Nary_Exp

Properties

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

Documentation

Represents a n-ary expression.

- The child elements represent the arguments.
- Attribute `op` represents the operator and shall be a nary_exp_op.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Nary_Exp
 op="nary_exp_op" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   Start Group: exp_group [1..*]
      Start Choice [1]
         <Unary_Exp> ... </Unary_Exp> [1]
         <Binary_Exp> ... </Binary_Exp> [1]
         <Ternary_Exp> ... </Ternary_Exp> [1]
         <Nary_Exp> ... </Nary_Exp> [1]
         <Boolean_Literal> ... </Boolean_Literal> [1]
         <Boolean_Exp> ... </Boolean_Exp> [1]
         <EmptySet> ... </EmptySet> [1]
         <EmptySeq> ... </EmptySeq> [1]
         <Id> ... </Id> [1]
         <Integer_Literal> ... </Integer_Literal> [1]
         <Quantified_Exp> ... </Quantified_Exp> [1]
         <Quantified_Set> ... </Quantified_Set> [1]
         <STRING_Literal> ... </STRING_Literal> [1]
         <Struct> ... </Struct> [1]
         <Record> ... </Record> [1]
         <Real_Literal> ... </Real_Literal> [1]
         <Record_Update> ... </Record_Update> [1]
         <Record_Field_Access> ... </Record_Field_Access> [1]
      End Choice
   End Group: exp_group
</Nary_Exp>
```

Schema Component Representation

```
<xs:element name="Nary_Exp">
   <xs:complexType>
      <xs:group ref="exp_group" minOccurs="1" maxOccurs="unbounded"/>
      <xs:attribute name="op" type="nary_exp_op" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Nary_Pred

Properties

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

Documentation

Represents a n-ary predicate.

- The child elements represent the arguments of the predicate.
- Attribute op represents the operator and shall be a nary_pred_op.

XML Instance Representation

```
<Nary_Pred
 op="nary_pred_op" [0..1]
>
   Start Group: pred_group [0..*]
      Start Choice [1]
         <Binary_Pred> ... </Binary_Pred> [1]
         <Exp_Comparison> ... </Exp_Comparison> [1]
         <Quantified_Pred> ... </Quantified_Pred> [1]
         <Unary_Pred> ... </Unary_Pred> [1]
         <Nary_Pred> ... </Nary_Pred> [1]
      End Choice
   End Group: pred_group
</Nary_Pred>
```

Schema Component Representation

```
<xs:element name="Nary_Pred">
   <xs:complexType>
      <xs:group ref="pred_group" minOccurs="0" maxOccurs="unbounded"/>
      <xs:attribute name="op" type="nary_pred_op"/>
   </xs:complexType>
</xs:element>
```

Element: Proof_Obligation

Properties

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

Documentation

This element represents a group of proof obligations.

- Child element Tag contains an informational description of the source of this group of proof obligations.
- Child elements Definition reference elements of the context.
- Child elements Hypothesis represent hypotheses that are common to all the proof obligations in this group.
- Child elements Local_Hyp represent hypotheses that are common to some of the proof obligations in this group.

- Finally, child elements Simple_Goal represent the proof obligations in this group.

XML Instance Representation

```
<Proof_Obligation
 goalHash="xs:nonNegativeInteger" [1]
>
   <Tag> xs:string </Tag> [1]
   <Definition> ... </Definition> [0..*]
   <Hypothesis> ... </Hypothesis> [0..*]
   <Local_Hyp> ... </Local_Hyp> [0..*]
   <Simple_Goal> ... </Simple_Goal> [0..*]
</Proof_Obligation>
```

Schema Component Representation

```
<xs:element name="Proof_Obligation">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Tag" type="xs:string" minOccurs="1" maxOccurs="1"/>
         <xs:element ref="Definition" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Hypothesis" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Local_Hyp" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Simple_Goal" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="goalHash" type="xs:nonNegativeInteger" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Proof_Obligations

Properties

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

Documentation

All proof obligations in a component reference sets of hypotheses originating from different clauses of the current component, and of components referenced from the current component. These references compose the *context* of the proof obligations.

So, in a POG file, the context is split into several sets of hypotheses, according to their origin in the source components. Such sets of hypotheses are represented by root child elements named Define.

Proof obligations are grouped according to their origin in the component. For instance, the following component clauses give rise to one group of proof obligations :

- for the initialisation;
- for each operation;
- for the well-definedness of each clause of the component containing expressions;

In a POG file, such groups are represented by root child elements named Proof_Obligation.

Finally, typing information of expression is represented in a root child element named TypeInfos.

XML Instance Representation

```
<Proof_Obligations
 version="version_type" [1]
>
   <Define> ... </Define> [0..*]
   <Proof_Obligation> ... </Proof_Obligation> [0..*]
   <TypeInfos> ... </TypeInfos> [0..1]
</Proof_Obligations>
```

Schema Component Representation

```
<xs:element name="Proof_Obligations">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Define" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="Proof_Obligation" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element ref="TypeInfos" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
      <xs:attribute name="version" type="version_type" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Proof_State

Properties

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

Documentation

Deprecated (to be removed).

XML Instance Representation

```
<Proof_State
 passList="xs:string" [1]
 methodList="xs:string" [1]
 proofState="xs:string" [1]
/>
```

Schema Component Representation

```
<xs:element name="Proof_State">
   <xs:complexType>
      <xs:attribute name="passList" type="xs:string" use="required"/>
      <xs:attribute name="methodList" type="xs:string" use="required"/>
      <xs:attribute name="proofState" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Quantified_Exp

Properties

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

Documentation

Represents a quantifying expression: either a lambda-expression, or a quantified sum or product, or a quantified union or intersection.

- Child `Variables` is a variables_type and represents the list of quantified variables.
- Child `Pred` is a predicate_type and represents the typing and otherwise constraining predicate.
- Child element `Body` is the quantified expression; it is of type expression_type.
- Attribute `type` represents the operator and shall be a quantified_exp_op.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Quantified_Exp
 type="quantified exp op" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   <Variables> variables type </Variables> [1]
   <Pred> predicate type </Pred> [1]
   <Body> expression type </Body> [1]
</Quantified_Exp>
```

Schema Component Representation

```
<xs:element name="Quantified_Exp">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Variables" type="variables type"/>
         <xs:element name="Pred" type="predicate type"/>
         <xs:element name="Body" type="expression type"/>
      </xs:sequence>
      <xs:attribute name="type" type="quantified exp op" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Quantified_Pred

Properties

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

Documentation

Represents a quantification predicate

- Child element `Variables` represents the list of quantified variables and is of type variables_type.
- Child element `Body` represents the quantified predicate; it is of type predicate_type.
- Attribute `type` represents the quantifier and shall be a quantified_pred_op.

XML Instance Representation

```
<Quantified_Pred
 type="quantified pred op" [1]
>
   <Variables> variables type </Variables> [1]
   <Body> predicate type </Body> [1]
</Quantified_Pred>
```

Schema Component Representation

```
<xs:element name="Quantified_Pred">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Variables" type="variables type"/>
         <xs:element name="Body" type="predicate type"/>
      </xs:sequence>
      <xs:attribute name="type" type="quantified pred op" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: Quantified_Set

Properties

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

Documentation

Represents a set defined in comprehension.

- Child `Variables` is a variables_type and represents variables appearing in the comprehension list.
- Child `Body` is a predicate_type and represents the predicate characterizing the set elements.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Quantified_Set
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   <Variables> variables_type </Variables> [1]
   <Body> predicate_type </Body> [1]
</Quantified_Set>
```

Schema Component Representation

```
<xs:element name="Quantified_Set">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Variables" type="variables_type"/>
         <xs:element name="Body" type="predicate_type"/>
      </xs:sequence>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Real_Literal

Properties

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

Documentation

Represents a real number literal expression.

- Attribute `value` is the represented literal and is a decimal (e.g., `value="3.1415"`).
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Real_Literal
 value="xs:decimal" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="Real_Literal">
   <xs:complexType>
      <xs:attribute name="value" type="xs:decimal" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Record

Properties

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

Documentation

Represents a record in extension, as described in [BLRM, §5.9].

- Children `Record_Item` are record_item_type elements and represent the different fields in the expression.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Record
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   <Record_Item> record_item_type </Record_Item> [1..*]
</Record>
```

Schema Component Representation

```
<xs:element name="Record">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Record_Item" type="record_item_type" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Record_Field_Access

Properties

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

Documentation

Represents the value of a field in a record.

- Child element represents the record.

XML Instance Representation

```
<Record_Field_Access
 typref="xs:integer" [1]
 label="xs:string" [1]
 tag="xs:string" [0..1]
>
   Start Choice [1]
      <Unary_Exp> ... </Unary_Exp> [1]
      <Binary_Exp> ... </Binary_Exp> [1]
      <Ternary_Exp> ... </Ternary_Exp> [1]
      <Nary_Exp> ... </Nary_Exp> [1]
      <Boolean_Literal> ... </Boolean_Literal> [1]
      <Boolean_Exp> ... </Boolean_Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer_Literal> ... </Integer_Literal> [1]
      <Quantified_Exp> ... </Quantified_Exp> [1]
      <Quantified_Set> ... </Quantified_Set> [1]
      <STRING_Literal> ... </STRING_Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real_Literal> ... </Real_Literal> [1]
      <Record_Update> ... </Record_Update> [1]
      <Record_Field_Access> ... </Record_Field_Access> [1]
   End Choice
</Record_Field_Access>
```

Schema Component Representation

```
<xs:element name="Record_Field_Access">
   <xs:complexType>
      <xs:group ref="exp_group" minOccurs="1" maxOccurs="1"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="label" type="xs:string" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Record_Update

Properties

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

Documentation

Represents an updated record. Such expressions are introduced during normalization of a "becomes equal" substitution where the target is the field of a record, to a "becomes equal" substitution where the target is a whole record.

For instance `point'xx := point'xx + 1` is normalized to `point := record_update(point, xx, point'xx + 1)`

- The first element represents the target record.
- The attribute 'label' represents the target field.
- The last element represents the source expression.
- Optional attribute `tag` represents the position of the expression(s) in the source code.

XML Instance Representation

```
<Record_Update
 label="xs:string" [1]
 tag="xs:string" [0..1]
>
   Start Choice [1]
      <Unary_Exp> ... </Unary_Exp> [1]
      <Binary_Exp> ... </Binary_Exp> [1]
      <Ternary_Exp> ... </Ternary_Exp> [1]
      <Nary_Exp> ... </Nary_Exp> [1]
      <Boolean_Literal> ... </Boolean_Literal> [1]
      <Boolean_Exp> ... </Boolean_Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer_Literal> ... </Integer_Literal> [1]
      <Quantified_Exp> ... </Quantified_Exp> [1]
      <Quantified_Set> ... </Quantified_Set> [1]
      <STRING_Literal> ... </STRING_Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real_Literal> ... </Real_Literal> [1]
      <Record_Update> ... </Record_Update> [1]
      <Record_Field_Access> ... </Record_Field_Access> [1]
   End Choice
   Start Choice [1]
      <Unary_Exp> ... </Unary_Exp> [1]
      <Binary_Exp> ... </Binary_Exp> [1]
      <Ternary_Exp> ... </Ternary_Exp> [1]
      <Nary_Exp> ... </Nary_Exp> [1]
      <Boolean_Literal> ... </Boolean_Literal> [1]
      <Boolean_Exp> ... </Boolean_Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer_Literal> ... </Integer_Literal> [1]
      <Quantified_Exp> ... </Quantified_Exp> [1]
      <Quantified_Set> ... </Quantified_Set> [1]
      <STRING_Literal> ... </STRING_Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real_Literal> ... </Real_Literal> [1]
      <Record_Update> ... </Record_Update> [1]
      <Record_Field_Access> ... </Record_Field_Access> [1]
   End Choice
</Record_Update>
```

Schema Component Representation

```
<xs:element name="Record_Update">
   <xs:complexType>
      <xs:sequence>
         <xs:group ref="exp_group"/>
         <xs:group ref="exp_group"/>
      </xs:sequence>
      <xs:attribute name="label" type="xs:string" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Ref_Hyp

Properties

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

Documentation

Represents the reference to a local hypothesis in a proof obligation. The value of attribute `num` identifies the referenced hypothesis.

XML Instance Representation

```
<Ref_Hyp
 num="xs:positiveInteger" [1]
/>
```

Schema Component Representation

```
<xs:element name="Ref_Hyp">
   <xs:complexType>
      <xs:attribute name="num" type="xs:positiveInteger" use="required"/>
   </xs:complexType>
</xs:element>
```

Element: STRING_Literal

Properties

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

Documentation

Represents a string literal expression.

- Attribute `value` is the represented literal and is a string.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<STRING_Literal
 value="xs:string" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
/>
```

Schema Component Representation

```
<xs:element name="STRING_Literal">
   <xs:complexType>
      <xs:attribute name="value" type="xs:string" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Set

Properties

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

Documentation

Represents the definition of a set (in the SETS clause of a component).

- Child element Id represents the set identifier.
- If there are no child elements Enumerated_Values, then it is an abstract set, otherwise it is an enumeration.

XML Instance Representation

```
<Set>
   <Id> ... </Id> [1]
   <Enumerated_Values> ... </Enumerated_Values> [0..1]
</Set>
```

Schema Component Representation

```
<xs:element name="Set">
   <xs:complexType>
      <xs:sequence>
         <xs:element ref="Id" minOccurs="1" maxOccurs="1"/>
         <xs:element ref="Enumerated_Values" minOccurs="0" maxOccurs="1"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Element: Simple_Goal

Properties

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

Documentation

Represents an individual proof obligation in a Proof_Obligation group.

- Child element Tag contains an informational text describing the role of the proof obligation.
- Child elements Ref_Hyp are references to Local_Hyp elements, representing hypotheses local to the current group.
- Child element Goal contains the goal predicate of the proof obligation.
- Child element Proof_State is obsolete.

XML Instance Representation

```
<Simple_Goal>
   <Tag> xs:string </Tag> [1]
   <Ref_Hyp> ... </Ref_Hyp> [0..*]
   <Goal> predicate_type </Goal> [0..1]
   <Proof_State> ... </Proof_State> [0..1]
</Simple_Goal>
```

Schema Component Representation

```
<xs:element name="Simple_Goal">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Tag" type="xs:string"/>
         <xs:element ref="Ref_Hyp" minOccurs="0" maxOccurs="unbounded"/>
         <xs:element name="Goal" type="predicate_type" minOccurs="0"/>
         <xs:element ref="Proof_State" minOccurs="0"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Element: Struct

Properties

Name	Struct
Type	Locally-defined complex type
Nillable	no

Abstract	no

Documentation

Represents a set of records, as described in [BLRM, §5.9].

- Children `Record_Item` are record_item_type elements and represent the different fields in the expression.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Struct
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   <Record_Item> record_item_type </Record_Item> [1..*]
</Struct>
```

Schema Component Representation

```
<xs:element name="Struct">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Record_Item" type="record_item_type" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Ternary_Exp

Properties

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

Documentation

Represents a ternary expression.

- Three child elements represent the arguments.
- Attribute `op` represents the operator and shall be a ternary_exp_op.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Ternary_Exp
 op="ternary_exp_op" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   Start Group: exp_group [3..3]
      Start Choice [1]
         <Unary_Exp> ... </Unary_Exp> [1]
         <Binary_Exp> ... </Binary_Exp> [1]
         <Ternary_Exp> ... </Ternary_Exp> [1]
         <Nary_Exp> ... </Nary_Exp> [1]
         <Boolean_Literal> ... </Boolean_Literal> [1]
         <Boolean_Exp> ... </Boolean_Exp> [1]
         <EmptySet> ... </EmptySet> [1]
         <EmptySeq> ... </EmptySeq> [1]
         <Id> ... </Id> [1]
         <Integer_Literal> ... </Integer_Literal> [1]
         <Quantified_Exp> ... </Quantified_Exp> [1]
         <Quantified_Set> ... </Quantified_Set> [1]
         <STRING_Literal> ... </STRING_Literal> [1]
         <Struct> ... </Struct> [1]
         <Record> ... </Record> [1]
         <Real_Literal> ... </Real_Literal> [1]
         <Record_Update> ... </Record_Update> [1]
         <Record_Field_Access> ... </Record_Field_Access> [1]
      End Choice
   End Group: exp_group
</Ternary_Exp>
```

Schema Component Representation

```
<xs:element name="Ternary_Exp">
   <xs:complexType>
      <xs:group ref="exp_group" minOccurs="3" maxOccurs="3"/>
      <xs:attribute name="op" type="ternary_exp_op" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: TypeInfos

Properties

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

Documentation

This is the element containing all the types of the expressions appearing in the proof obligations.

Each child `Type` is a typeinfos_type element and represents a B type.

XML Instance Representation

```
<TypeInfos>
   <Type> typeinfos_type </Type> [0..*]
</TypeInfos>
```

Schema Component Representation

```
<xs:element name="TypeInfos">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="Type" type="typeinfos_type" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
</xs:element>
```

Element: Unary_Exp

Properties

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

Documentation

Represents a unary expression.

- The single child element represents the argument.
- Attribute `op` represents the operator and shall be a unary_exp_op.
- Attribute `typref` is the index of the type representation in the TypeInfos element of the document.
- Optional attribute `tag` represents a position or a list of positions in the source bxml. It can be used to trace the origin of the expression.

XML Instance Representation

```
<Unary_Exp
 op="unary_exp_op" [1]
 typref="xs:integer" [1]
 tag="xs:string" [0..1]
>
   Start Choice [1]
      <Unary_Exp> ... </Unary_Exp> [1]
      <Binary_Exp> ... </Binary_Exp> [1]
      <Ternary_Exp> ... </Ternary_Exp> [1]
      <Nary_Exp> ... </Nary_Exp> [1]
      <Boolean_Literal> ... </Boolean_Literal> [1]
      <Boolean_Exp> ... </Boolean_Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer_Literal> ... </Integer_Literal> [1]
      <Quantified_Exp> ... </Quantified_Exp> [1]
      <Quantified_Set> ... </Quantified_Set> [1]
      <STRING_Literal> ... </STRING_Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real_Literal> ... </Real_Literal> [1]
      <Record_Update> ... </Record_Update> [1]
      <Record_Field_Access> ... </Record_Field_Access> [1]
   End Choice
</Unary_Exp>
```

Schema Component Representation

```
<xs:element name="Unary_Exp">
   <xs:complexType>
      <xs:group ref="exp_group"/>
      <xs:attribute name="op" type="unary_exp_op" use="required"/>
      <xs:attribute name="typref" type="xs:integer" use="required"/>
      <xs:attribute name="tag" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

Element: Unary_Pred

Properties

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

Documentation

Represents a unary predicate.

- The single child element represent the arguments to the predicate.
- Attribute op represents the operator and shall be a unary_pred_op.

XML Instance Representation

```
<Unary_Pred
 op="unary_pred_op" [1]
>
   Start Choice [1]
      <Binary_Pred> ... </Binary_Pred> [1]
      <Exp_Comparison> ... </Exp_Comparison> [1]
      <Quantified_Pred> ... </Quantified_Pred> [1]
      <Unary_Pred> ... </Unary_Pred> [1]
      <Nary_Pred> ... </Nary_Pred> [1]
   End Choice
</Unary_Pred>
```

Schema Component Representation

```
<xs:element name="Unary_Pred">
   <xs:complexType>
      <xs:group ref="pred_group"/>
      <xs:attribute name="op" type="unary_pred_op" use="required"/>
   </xs:complexType>
</xs:element>
```

Global Definitions

Complex Type: expression_type

Type hierarchy

Super-types:	None
Sub-types:	None

Properties

Name	expression_type
Abstract	no

Documentation

No documentation provided.

XML Instance Representation

```
<...>
   Start Choice [1]
      <Unary_Exp> ... </Unary_Exp> [1]
      <Binary_Exp> ... </Binary_Exp> [1]
      <Ternary_Exp> ... </Ternary_Exp> [1]
      <Nary_Exp> ... </Nary_Exp> [1]
      <Boolean_Literal> ... </Boolean_Literal> [1]
      <Boolean_Exp> ... </Boolean_Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer_Literal> ... </Integer_Literal> [1]
      <Quantified_Exp> ... </Quantified_Exp> [1]
      <Quantified_Set> ... </Quantified_Set> [1]
      <STRING_Literal> ... </STRING_Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real_Literal> ... </Real_Literal> [1]
      <Record_Update> ... </Record_Update> [1]
      <Record_Field_Access> ... </Record_Field_Access> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="expression_type">
   <xs:group ref="exp_group"/>
</xs:complexType>
```

Complex Type: predicate_type

Type hierarchy

Super-types:	None
Sub-types:	None

Properties

Name	predicate_type
Abstract	no

Documentation

No documentation provided.

XML Instance Representation

```
<...>
   Start Choice [1]
      <Binary_Pred> ... </Binary_Pred> [1]
      <Exp_Comparison> ... </Exp_Comparison> [1]
      <Quantified_Pred> ... </Quantified_Pred> [1]
      <Unary_Pred> ... </Unary_Pred> [1]
      <Nary_Pred> ... </Nary_Pred> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="predicate_type">
   <xs:group ref="pred_group"/>
</xs:complexType>
```

Complex Type: record_item_type

Type hierarchy

Super-types:	None
Sub-types:	None

Properties

Name	record_item_type
Abstract	no

Documentation

Represents a field in a record or a set of record expression (BLRM,§5.9).

- Attribute `label` is the field identifier.
- The child represents the field expression and is an element in exp_group.

XML Instance Representation

```
<...
 label="xs:string" [1]
>
   Start Choice [1]
      <Unary Exp> ... </Unary Exp> [1]
      <Binary Exp> ... </Binary Exp> [1]
      <Ternary Exp> ... </Ternary Exp> [1]
      <Nary Exp> ... </Nary Exp> [1]
      <Boolean Literal> ... </Boolean Literal> [1]
      <Boolean Exp> ... </Boolean Exp> [1]
      <EmptySet> ... </EmptySet> [1]
      <EmptySeq> ... </EmptySeq> [1]
      <Id> ... </Id> [1]
      <Integer Literal> ... </Integer Literal> [1]
      <Quantified Exp> ... </Quantified Exp> [1]
      <Quantified Set> ... </Quantified Set> [1]
      <STRING Literal> ... </STRING Literal> [1]
      <Struct> ... </Struct> [1]
      <Record> ... </Record> [1]
      <Real Literal> ... </Real Literal> [1]
      <Record Update> ... </Record Update> [1]
      <Record Field Access> ... </Record Field Access> [1]
   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="record_item_type">
   <xs:group ref="exp group"/>
   <xs:attribute name="label" type="xs:string" use="required"/>
</xs:complexType>
```

Complex Type: typeinfos_type

Type hierarchy

Super-types:	None
Sub-types:	None

Properties

Name	typeinfos_type
Abstract	no

Documentation

The elements representing entries in the TypeInfos of the document.

Integer attribute `id` identifies uniquely the type expression and is used in type references for elements representing expressions.

XML Instance Representation

```
<...
 id="xs:integer" [1]
>
   Start Choice [1]
      <Binary_Exp
       op="*" [1]
      > [1]
         Circular model group reference: type group [2..2]
      </Binary_Exp>
      <Id
       value="xs:string" [1]
/>  [1]

      <Unary_Exp
       op="POW" [1]
      > [1]
         Circular model group reference: type group [1]
      </Unary_Exp>
      <Struct      > [1]
         <Record_Item
          label="xs:string" [1]
         > [1..*]
            Circular model group reference: type group [1]
         </Record_Item>
      </Struct>
      <Generic_Type/>  [1]

   End Choice
</...>
```

Schema Component Representation

```
<xs:complexType name="typeinfos_type">
   <xs:group ref="type group"/>
   <xs:attribute name="id" type="xs:integer" use="required"/>
</xs:complexType>
```

Complex Type: variables_type

Type hierarchy

Super-types:	None
Sub-types:	None

Properties

Name	variables_type
Abstract	no

Documentation

Represents a list of quantified identifiers, each being a child element Id.

XML Instance Representation

```
<...>
   <Id> ... </Id> [0..*]
</...>
```

Schema Component Representation

```
<xs:complexType name="variables_type">
   <xs:sequence>
      <xs:element ref="Id" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
</xs:complexType>
```

Model Group: exp_group

Properties

Name	exp_group

Documentation

The different types of elements representing expressions.

XML Instance Representation

```
Start Choice [1]
   <Unary Exp> ... </Unary Exp> [1]
   <Binary Exp> ... </Binary Exp> [1]
   <Ternary Exp> ... </Ternary Exp> [1]
   <Nary Exp> ... </Nary Exp> [1]
   <Boolean Literal> ... </Boolean Literal> [1]
   <Boolean Exp> ... </Boolean Exp> [1]
   <EmptySet> ... </EmptySet> [1]
   <EmptySeq> ... </EmptySeq> [1]
   <Id> ... </Id> [1]
   <Integer Literal> ... </Integer Literal> [1]
   <Quantified Exp> ... </Quantified Exp> [1]
   <Quantified Set> ... </Quantified Set> [1]
   <STRING Literal> ... </STRING Literal> [1]
   <Struct> ... </Struct> [1]
   <Record> ... </Record> [1]
   <Real Literal> ... </Real Literal> [1]
   <Record Update> ... </Record Update> [1]
   <Record Field Access> ... </Record Field Access> [1]
End Choice
```

Schema Component Representation

```
<xs:group name="exp_group">
   <xs:choice>
      <xs:element ref="Unary Exp"/>
      <xs:element ref="Binary Exp"/>
      <xs:element ref="Ternary Exp"/>
      <xs:element ref="Nary Exp"/>
      <xs:element ref="Boolean Literal"/>
      <xs:element ref="Boolean Exp"/>
      <xs:element ref="EmptySet"/>
      <xs:element ref="EmptySeq"/>
      <xs:element ref="Id"/>
      <xs:element ref="Integer Literal"/>
      <xs:element ref="Quantified Exp"/>
      <xs:element ref="Quantified Set"/>
      <xs:element ref="STRING Literal"/>
      <xs:element ref="Struct"/>
      <xs:element ref="Record"/>
      <xs:element ref="Real Literal"/>
      <xs:element ref="Record Update"/>
      <xs:element ref="Record Field Access"/>
   </xs:choice>
</xs:group>
```

Model Group: pred_group

Properties

Name	pred_group

Documentation

The different types of elements representing predicates.

XML Instance Representation

```
Start Choice [1]
   <Binary Pred> ... </Binary Pred> [1]
   <Exp Comparison> ... </Exp Comparison> [1]
   <Quantified Pred> ... </Quantified Pred> [1]
   <Unary Pred> ... </Unary Pred> [1]
   <Nary Pred> ... </Nary Pred> [1]
End Choice
```

Schema Component Representation

```
<xs:group name="pred_group">
   <xs:choice>
      <xs:element ref="Binary_Pred"/>
      <xs:element ref="Exp_Comparison"/>
      <xs:element ref="Quantified_Pred"/>
      <xs:element ref="Unary_Pred"/>
      <xs:element ref="Nary_Pred"/>
   </xs:choice>
</xs:group>
```

Model Group: type_group

Properties

Name	type_group

Documentation

Represents a B type (BLRM,§3.2).

The following representations exist:

- `Binary_Exp` to represent a Cartesian product type.
- `Unary_Exp` to represent a powerset type.
- `Struct` to represent a record type.
- `Id` to represent either a predefined type or a deferred set.
- `Generic_Type` for expression that could not be given a B type (e.g. an empty set expression may have type `<Unary_Exp op="POW"><Generic_Type/></Unary_Exp>`).

XML Instance Representation

```
Start Choice [1]
   <Binary_Exp
    op="*" [1]
   > [1]
      Circular model group reference: type_group [2..2]
   </Binary_Exp>
   <Id
    value="xs:string" [1]
/>  [1]

   <Unary_Exp
    op="POW" [1]
   > [1]
      Circular model group reference: type_group [1]
   </Unary_Exp>
   <Struct   > [1]
      <Record_Item
       label="xs:string" [1]
      > [1..*]
         Circular model group reference: type_group [1]
      </Record_Item>
   </Struct>
   <Generic_Type/>  [1]

End Choice
```

Schema Component Representation

```
<xs:group name="type_group">
   <xs:choice>
      <xs:element name="Binary_Exp">
         <xs:complexType>
            <xs:group ref="type_group" minOccurs="2" maxOccurs="2"/>
            <xs:attribute name="op" type="xs:string" use="required" fixed="*"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Id">
         <xs:complexType>
            <xs:attribute name="value" type="xs:string" use="required"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Unary_Exp">
         <xs:complexType>
            <xs:group ref="type_group"/>
            <xs:attribute name="op" type="xs:string" use="required" fixed="POW"/>
         </xs:complexType>
      </xs:element>
      <xs:element name="Struct">
         <xs:complexType>
            <xs:sequence>
               <xs:element name="Record_Item" minOccurs="1" maxOccurs="unbounded">
                  <xs:complexType>
                     <xs:group ref="type_group"/>
                     <xs:attribute name="label" type="xs:string" use="required"/>
                  </xs:complexType>
               </xs:element>
            </xs:sequence>
         </xs:complexType>
      </xs:element>
      <xs:element name="Generic_Type">
         <xs:complexType/>
      </xs:element>
   </xs:choice>
</xs:group>
```

Simple Type: binary_exp_op

Type hierarchy

Super-types:	xs:string < **binary_exp_op** (by restriction)
Sub-types:	None

Properties

Name binary_exp_op

Content

- Base XSD Type: string
- *value* comes from list: {','|'*'|'*i'|'*r'|'*f'|'*s'|'**'|'**i'|'**r'|'+'|'+i'|'+r'|'+f'|'+->'|'+->>'|'-'|'-i'|'-r'|'-f'|'-s'|'-->'|'-->>'|'->'|'..'|'/'|'/i'|'/r'|'/f'|'/\'|'/|\'|';'|'<+'|'<->'|'<-'|'<<|'|'<|'|'>+>'|'>->'|'>+>>'|'>->>'|'><'|'||'|'\/'|'\|/'|'^'|'mod'|'|->'|'|>'|'|>>'|'['|'('|'<''|'prj1'|'prj2'|'iterate'|'const'|'rank'|'father'|'subtree'|'arity'}

Documentation

Represents the possible binary expression operators. The meaning of the operators is described in (BLRM,§A).

Overloaded B operators are resolved :

- `*` is resolved to `"*s"` (Cartesian product), `"*i"` (multiplication of integers), `"*r"` (multiplication of real numbers), `"*f"` (multiplication of floating point numbers).
- `**` is resolved to `"*s"` (Cartesian product), `"**i"` (exponentiation of integers), `"**r"` (exponentiation of a real numbers).
- `"+"` is resolved to `"+i"` (addition of integers), `"+r"` (addition of real numbers), `"+f"` (addition of floating point numbers).
- `-` is resolved to `"-s"` (set difference), `"-i"` (subtraction of integers), `"-r"` (subtraction of real numbers), `"-f"` (subtraction of floating point numbers).
- `/` is resolved to `"/i"` (division quotient of integers), `"/r"` (division of real numbers), `"/f"` (division of floating point numbers).

Schema Component Representation

```
<xs:simpleType name="binary_exp_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value=","/>
      <xs:enumeration value="*"/>
      <xs:enumeration value="*i"/>
      <xs:enumeration value="*r"/>
      <xs:enumeration value="*f"/>
      <xs:enumeration value="*s"/>
      <xs:enumeration value="**"/>
      <xs:enumeration value="**i"/>
      <xs:enumeration value="**r"/>
      <xs:enumeration value="+"/>
      <xs:enumeration value="+i"/>
      <xs:enumeration value="+r"/>
      <xs:enumeration value="+f"/>
      <xs:enumeration value="+->"/>
      <xs:enumeration value="+->>"/>
      <xs:enumeration value="-"/>
      <xs:enumeration value="-i"/>
      <xs:enumeration value="-r"/>
      <xs:enumeration value="-f"/>
      <xs:enumeration value="-s"/>
      <xs:enumeration value="-->"/>
      <xs:enumeration value="-->>"/>
      <xs:enumeration value="->"/>
      <xs:enumeration value=".."/>
      <xs:enumeration value="/"/>
      <xs:enumeration value="/i"/>
      <xs:enumeration value="/r"/>
      <xs:enumeration value="/f"/>
      <xs:enumeration value="/\"/>
      <xs:enumeration value="/|\"/>
      <xs:enumeration value=";"/>
      <xs:enumeration value="<+"/>
      <xs:enumeration value="<->"/>
      <xs:enumeration value="<-"/>
      <xs:enumeration value="<<|"/>
      <xs:enumeration value="<|"/>
      <xs:enumeration value=">+>"/>
      <xs:enumeration value=">->"/>
      <xs:enumeration value=">+>>"/>
      <xs:enumeration value=">->>"/>
      <xs:enumeration value="><"/>
      <xs:enumeration value="||"/>
      <xs:enumeration value="\/"/>
      <xs:enumeration value="\|/"/>
      <xs:enumeration value="^"/>
      <xs:enumeration value="mod"/>
      <xs:enumeration value="|->"/>
      <xs:enumeration value="|>"/>
      <xs:enumeration value="|>>"/>
      <xs:enumeration value="["/>
      <xs:enumeration value="("/>
      <xs:enumeration value="<'"/>
      <xs:enumeration value="prj1"/>
      <xs:enumeration value="prj2"/>
      <xs:enumeration value="iterate"/>
      <xs:enumeration value="const"/>
      <xs:enumeration value="rank"/>
      <xs:enumeration value="father"/>
      <xs:enumeration value="subtree"/>
      <xs:enumeration value="arity"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: binary_pred_op

Type hierarchy

Super-types:	xs:string < **binary_pred_op** (by restriction)
Sub-types:	None

Properties

Name	binary_pred_op
Content	• Base XSD Type: string • *value* comes from list: {'=>'|'<=>'}

Documentation

Represents the possible values of a binary predicate operator. They correspond to implication (`"=>"`) and equivalence (`"<=>"`) .

Schema Component Representation

```
<xs:simpleType name="binary_pred_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="=>"/>
      <xs:enumeration value="<=>"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: boolean_literal_type

Type hierarchy

Super-types:	xs:string < **boolean_literal_type** (by restriction)
Sub-types:	None

Properties

Name	boolean_literal_type
Content	• Base XSD Type: string • *value* comes from list: {'TRUE'|'FALSE'}

Documentation

Restricts the possible values of a Boolean literal.

Schema Component Representation

```
<xs:simpleType name="boolean_literal_type">
   <xs:restriction base="xs:string">
      <xs:enumeration value="TRUE"/>
      <xs:enumeration value="FALSE"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: comparison_op

Type hierarchy

Super-types:	xs:string < **comparison_op** (by restriction)
Sub-types:	None

Properties

<table>
<tr><td><b>Name</b></td><td>comparison_op</td></tr>
<tr><td><b>Content</b></td><td>• Base XSD Type: string
• <i>value</i> comes from list: {':'|'/:'|'<:'|'/<:'|'<<:'|'/<<:'|'='|'/='|'>=i'|'>i'|'<i'|'<=i'|'>=r'|'>r'|'<r'|'<=r'|'>=f'|'>f'|'<f'|'<=f'}</td></tr>
</table>

Documentation

Represents the possible values of a comparison operator. The meaning of the operators is described in (BLRM,§A).

Some overloaded B operators are resolved :

- `>` (greater than) is resolved to `">i"` (integers), `">r"` (real numbers), `">f"` (floating point numbers).
- `>=` (greater than or equal to) is resolved to `">=i"` (integers), `">=r"` (real numbers), `">=f"` (floating point numbers).
- `<` (lower than) is resolved to `"<i"` (integers), `"<r"` (real numbers), `"<f"` (floating point numbers).
- `<=` (lower than or equal to) is resolved to `"<=i"` (integers), `"<=r"` (real numbers), `"<=f"` (floating point numbers).

Schema Component Representation

```
<xs:simpleType name="comparison_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value=":"/>
      <xs:enumeration value="/:"/>
      <xs:enumeration value="<:"/>
      <xs:enumeration value="/<:"/>
      <xs:enumeration value="<<:"/>
      <xs:enumeration value="/<<:"/>
      <xs:enumeration value="="/>
      <xs:enumeration value="/="/>
          <-- integer comparison -->      <xs:enumeration value=">=i"/>
      <xs:enumeration value=">i"/>
      <xs:enumeration value="<i"/>
      <xs:enumeration value="<=i"/>
          <-- real comparison -->      <xs:enumeration value=">=r"/>
      <xs:enumeration value=">r"/>
      <xs:enumeration value="<r"/>
      <xs:enumeration value="<=r"/>
          <-- float comparison -->      <xs:enumeration value=">=f"/>
      <xs:enumeration value=">f"/>
      <xs:enumeration value="<f"/>
      <xs:enumeration value="<=f"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: nary_exp_op

Type hierarchy

Super-types:	xs:string < **nary_exp_op** (by restriction)
Sub-types:	None

Properties

Name	nary_exp_op
Content	• Base XSD Type: string • *value* comes from list: {'['|'{'}

Documentation

Represents the possible n-ary expression operators:

- `"["` is used to represent a sequence literal.
- `"{"` is used to represent a set literal defined in extension.

Schema Component Representation

```
<xs:simpleType name="nary_exp_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="["/>
      <xs:enumeration value="{"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: nary_pred_op

Type hierarchy

Super-types:	xs:string < **nary_pred_op** (by restriction)
Sub-types:	None

Properties

Name	nary_pred_op
Content	• Base XSD Type: string • *value* comes from list: {'&'|'or'}

Documentation

Represents the possible n-ary predicate operators : conjunction (`"&"`) and disjunction (`"or"`).

Schema Component Representation

```
<xs:simpleType name="nary_pred_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="&"/>
      <xs:enumeration value="or"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: quantified_exp_op

Type hierarchy

Super-types:	xs:string < **quantified_exp_op** (by restriction)
Sub-types:	None

Properties

Name	quantified_exp_op
Content	• Base XSD Type: string • *value* comes from list: {'%'|'SIGMA'|'iSIGMA'|'rSIGMA'|'PI'|'iPI'|'rPI'|'INTER'|'UNION'}

Documentation

Represents the possible expression quantifiers. The meaning of the quantifiers is described in (BLRM,§A).

Overloaded B operators are resolved :

- `SIGMA` is resolved to `"iSIGMA"` (sum over integers), `"rSIGMA"` (sum over real numbers).
- `PI` is resolved to `"iPI"` (product over integers), `"rPI"` (product over real numbers).

Schema Component Representation

```
<xs:simpleType name="quantified_exp_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="%"/>
      <xs:enumeration value="SIGMA"/>
      <xs:enumeration value="iSIGMA"/>
      <xs:enumeration value="rSIGMA"/>
      <xs:enumeration value="PI"/>
      <xs:enumeration value="iPI"/>
      <xs:enumeration value="rPI"/>
      <xs:enumeration value="INTER"/>
      <xs:enumeration value="UNION"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: quantified_pred_op

Type hierarchy

Super-types:	xs:string < **quantified_pred_op** (by restriction)
Sub-types:	None

Properties

Name	quantified_pred_op
Content	• Base XSD Type: string • *value* comes from list: {'!'|'#'}

Documentation

Represents the possible quantifiers: `"!"` for universal and `"#"` for existential.

Schema Component Representation

```
<xs:simpleType name="quantified_pred_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="!"/>
      <xs:enumeration value="#"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: ternary_exp_op

Type hierarchy

Super-types:	xs:string < **ternary_exp_op** (by restriction)
Sub-types:	None

Properties

Name	ternary_exp_op
Content	• Base XSD Type: string • *value* comes from list: {'son'|'bin'}

Documentation

Represents the possible ternary expression operators. The meaning of the operators is described in (BLRM,§A).

Schema Component Representation

```
<xs:simpleType name="ternary_exp_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="son"/>
      <xs:enumeration value="bin"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: unary_exp_op

Type hierarchy

Super-types:	xs:string < **unary_exp_op** (by restriction)
Sub-types:	None

Properties

Name	unary_exp_op
Content	• Base XSD Type: string • *value* comes from list: {'max'|'imax'|'rmax'|'min'|'imin'|'rmin'|'card'|'dom'|'ran'|'POW'|'POW1'|'FIN'|'FIN1'|'union'|'inter'|'seq'|'seq1'|'iseq'|'iseq1'|'-'|'-i'|'-r'|'~'|'size'|'perm'|'first'|'last'|'id'|'closure'|'closure1'|'tail'|'front'|'rev'|'conc'|'succ'|'pred'|'rel'|'fnc'|'real'|'floor'|'ceiling'|'tree'|'btree'|'top'|'sons'|'prefix'|'postfix'|'siz

Documentation

Represents the possible unary expression operators. The meaning of the operators is described in (BLRM,§A).

Overloaded B operators are resolved :

- `max` is resolved to `"imax"` (maximum of a set of integers), `"rmax"` (maximum of a set of real numbers).
- `min` is resolved to `"imin"` (minimum of a set of integers), `"rmin"` (minimum of a set of real numbers).
- `-` (unary minus) is resolved to `"-i"` (unary minus over integers), `"-r"` (unary minus over real numbers).

Schema Component Representation

```
<xs:simpleType name="unary_exp_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="max"/>
      <xs:enumeration value="imax"/>
      <xs:enumeration value="rmax"/>
      <xs:enumeration value="min"/>
      <xs:enumeration value="imin"/>
      <xs:enumeration value="rmin"/>
      <xs:enumeration value="card"/>
      <xs:enumeration value="dom"/>
      <xs:enumeration value="ran"/>
      <xs:enumeration value="POW"/>
      <xs:enumeration value="POW1"/>
      <xs:enumeration value="FIN"/>
      <xs:enumeration value="FIN1"/>
      <xs:enumeration value="union"/>
      <xs:enumeration value="inter"/>
      <xs:enumeration value="seq"/>
      <xs:enumeration value="seq1"/>
      <xs:enumeration value="iseq"/>
      <xs:enumeration value="iseq1"/>
      <xs:enumeration value="-"/>
      <xs:enumeration value="-i"/>
      <xs:enumeration value="-r"/>
      <xs:enumeration value="~"/>
      <xs:enumeration value="size"/>
      <xs:enumeration value="perm"/>
      <xs:enumeration value="first"/>
      <xs:enumeration value="last"/>
      <xs:enumeration value="id"/>
      <xs:enumeration value="closure"/>
      <xs:enumeration value="closure1"/>
      <xs:enumeration value="tail"/>
      <xs:enumeration value="front"/>
      <xs:enumeration value="rev"/>
      <xs:enumeration value="conc"/>
      <xs:enumeration value="succ"/>
      <xs:enumeration value="pred"/>
      <xs:enumeration value="rel"/>
      <xs:enumeration value="fnc"/>
      <xs:enumeration value="real"/>
      <xs:enumeration value="floor"/>
      <xs:enumeration value="ceiling"/>
      <xs:enumeration value="tree"/>
      <xs:enumeration value="btree"/>
      <xs:enumeration value="top"/>
      <xs:enumeration value="sons"/>
      <xs:enumeration value="prefix"/>
      <xs:enumeration value="postfix"/>
      <xs:enumeration value="sizet"/>
      <xs:enumeration value="mirror"/>
      <xs:enumeration value="left"/>
      <xs:enumeration value="right"/>
      <xs:enumeration value="infix"/>
      <xs:enumeration value="bin"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: unary_pred_op

Type hierarchy

Super-types:	xs:string < **unary_pred_op** (by restriction)
Sub-types:	None

Properties

Name	unary_pred_op
Content	• Base XSD Type: string • *value* comes from list: {'not'}

Documentation

Represents the possible operators: `"not"` (for negation) is the only possibility.

Schema Component Representation

```
<xs:simpleType name="unary_pred_op">
   <xs:restriction base="xs:string">
      <xs:enumeration value="not"/>
   </xs:restriction>
</xs:simpleType>
```

Simple Type: version_type

Type hierarchy

Super-types:	xs:string < **version_type** (by restriction)
Sub-types:	None

Properties

Name	version_type
Content	• Base XSD Type: string • *value* comes from list: {'1.0'}

Documentation

Represents the possible values for the `version` attribute for the root element. Currently a unique value is possible: `"1.0"` . When the format evolves, the new versions will be added there.

Schema Component Representation

```
<xs:simpleType name="version_type">
   <xs:restriction base="xs:string">
      <xs:enumeration value="1.0"/>
   </xs:restriction>
</xs:simpleType>
```

Glossary

Abstract (Applies to complex type definitions and element declarations). An abstract element or complex type cannot used to validate an element instance. If there is a reference to an abstract element, only element declarations that can substitute the abstract element can be used to validate the instance. For references to abstract type definitions, only derived types can be used.

All Model Group Child elements can be provided *in any order* in instances. See: http://www.w3.org/TR/xmlschema-1/#element-all (http://www.w3.org/TR/xmlschema-1/#element-all).

Choice Model Group *Only one* from the list of child elements and model groups can be provided in instances. See: http://www.w3.org/TR/xmlschema-1/#element-choice (http://www.w3.org/TR/xmlschema-1/#element-choice).

Collapse Whitespace Policy Replace tab, line feed, and carriage return characters with space character (Unicode character 32). Then, collapse contiguous sequences of space characters into single space character, and remove leading and trailing space characters.

Disallowed Substitutions (Applies to element declarations). If *substitution* is specified, then substitution group members cannot be used in place of the given element declaration to validate element instances. If *derivation methods*, e.g. extension, restriction, are specified, then the given element declaration will not validate element instances that have types derived from the element declaration's type using the specified derivation methods. Normally, element instances can override their declaration's type by specifying an `xsi:type` attribute.

Key Constraint Like Uniqueness Constraint, but additionally requires that the specified value(s) must be provided. See: http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions (http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions).

Key Reference Constraint Ensures that the specified value(s) must match value(s) from a Key Constraint or Uniqueness Constraint. See: http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions (http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions).

Model Group Groups together element content, specifying the order in which the element content can occur and the number of times the group of element content may be repeated. See: http://www.w3.org/TR/xmlschema-1/#Model_Groups (http://www.w3.org/TR/xmlschema-1/#Model_Groups).

Nillable (Applies to element declarations). If an element declaration is nillable, instances can use the `xsi:nil` attribute. The `xsi:nil` attribute is the boolean attribute, *nil*, from the *http://www.w3.org/2001/XMLSchema-instance* namespace. If an element instance has an `xsi:nil` attribute set to true, it can be left empty, even though its element declaration may have required content.

Notation A notation is used to identify the format of a piece of data. Values of elements and attributes that are of type, NOTATION, must come from the names of declared notations. See: http://www.w3.org/TR/xmlschema-1/#cNotation_Declarations (http://www.w3.org/TR/xmlschema-1/#cNotation_Declarations).

Preserve Whitespace Policy Preserve whitespaces exactly as they appear in instances.

Prohibited Derivations (Applies to type definitions). Derivation methods that cannot be used to create sub-types from a given type definition.

Prohibited Substitutions (Applies to complex type definitions). Prevents sub-types that have been derived using the specified derivation methods from validating element instances in place of the given type definition.

Replace Whitespace Policy Replace tab, line feed, and carriage return characters with space character (Unicode character 32).

Sequence Model Group Child elements and model groups must be provided *in the specified order* in instances. See: http://www.w3.org/TR/xmlschema-1/#element-sequence (http://www.w3.org/TR/xmlschema-1/#element-sequence).

Substitution Group Elements that are *members* of a substitution group can be used wherever the *head* element of the substitution group is referenced.

Substitution Group Exclusions (Applies to element declarations). Prohibits element declarations from nominating themselves as being able to substitute a given element declaration, if they have types that are derived from the original element's type using the specified derivation methods.

Target Namespace The target namespace identifies the namespace that components in this schema belongs to. If no target namespace is provided, then the schema components do not belong to any namespace.

Uniqueness Constraint Ensures uniqueness of an element/attribute value, or a combination of values, within a specified scope. See: http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions (http://www.w3.org/TR/xmlschema-1/#cIdentity-constraint_Definitions).

Generated by xs3p-ch (http://github.com/bitfehler/xs3p) (fork of xs3p (http://xml.fiforms.org/xs3p/)) . Last Modified: 09/20/2019 10:05:40