W3C Editor's Draft

RIF Datatypes and Built-Ins 1.0

W3C Editor's Draft 18 December 2008

This version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-dtb-20081218/
Latest editor's draft:
http://www.w3.org/2005/rules/wg/draft/rif-dtb/
Previous version:
http://www.w3.org/2005/rules/wg/draft/ED-rif-dtb-20081125/ (color-coded diff)
Editors:
Axel Polleres, DERI
Harold Boley, National Research Council Canada
Michael Kifer, State University of New York at Stony Brook

This document is also available in these non-normative formats: PDF version.

Abstract

Status of this Document

May Be Superseded

This section describes the status of this document at the time of its publication. Other documents may supersede this document. A list of current W3C publications and the latest revision of this technical report can be found in the W3C technical reports index at http://www.w3.org/TR/.

Set of Documents

This document is being published as one of a set of 5 documents:

1. RIF Use Cases and Requirements

W3C Editor's Draft

2. RIF Core
3. RIF Datatypes and Built-Ins 1.0 (this document)
4. RIF Production Rule Dialect
5. RIF Test Cases

Please Comment By 23 January 2009

The Rule Interchange Format (RIF) Working Group seeks public feedback on these Working Drafts. Please send your comments to public-rif-comments@w3.org (public archive). If possible, please offer specific changes to the text that would address your concern. You may also wish to check the Wiki Version of this document for internal-review comments and changes being drafted which may address your concerns.

No Endorsement

Publication as a Working Draft does not imply endorsement by the W3C Membership. This is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to cite this document as other than work in progress.

Patents

This document was produced by a group operating under the 5 February 2004 W3C Patent Policy. W3C maintains a public list of any patent disclosures made in connection with the deliverables of the group; that page also includes instructions for disclosing a patent. An individual who has actual knowledge of a patent which the individual believes contains Essential Claim(s) must disclose the information in accordance with section 6 of the W3C Patent Policy.

This document, developed by the Rule Interchange Format (RIF) Working Group, specifies a list of primitive datatypes, built-in functions and built-in predicates expected to be supported by RIF dialects such as the RIF Basic Logic Dialect. Each dialect supporting a superset or subset of the primitive datatypes, built-in functions and built-in predicates defined here shall specify these additions or restrictions. Some of the datatypes are adopted from [XML-SCHEMA2]. A large part of the definitions of the listed functions and operators are adopted from [XPath-Functions]. The `rdf:text` datatype as well as functions and operators associated with that datatype are adopted from [RDF-TEXT].

W3C Editor's Draft

Contents

1 Constants, Symbol Spaces, and Datatypes

1.1 Constants and Symbol Spaces

Each constant (that is, each non-keyword symbol) in RIF belongs to a particular symbol space. A constant in a particular RIF symbol space has the following presentation syntax:

`"`*`literal`*`"^^<`*`symbolSpaceIri`*`>`

where *`literal`* is called the ***lexical part*** of the symbol, and *`symbolSpaceIri`* is an (absolute or relative) IRI identifying the ***symbol space***. Here `literal` is a Unicode string that must be an element in the lexical space of the symbol space identified by the IRI *symbolSpaceIri*.

1.2 The Base and Prefix Directives

Since IRI typically require long strings of characters, many Web languages have special provisions for abbreviating these strings. One popular technique is called *compact URI* [CURIE], and RIF uses a similar technique by allowing RIF documents to have the directives `Base` and `Prefix`.

- A ***base directive*** has the form `Base(iri)`, where `iri` is a unicode string in the form of an IRI [RFC-3987].

  The `Base` directive defines a syntactic shortcut for expanding relative IRIs into full IRIs.

- A ***prefix directive*** has the form `Prefix(p v)`, where `p` is called a ***prefix*** and `v` is its ***expansion***. A prefix is an alphanumeric string an expansion is a string that forms an IRI. (An alphanumeric string is a sequence of ASCII characters, where each character is a letter, a digit, or an underscore "_", and the first character is a letter.)

  The basic idea is that in certain contexts prefixes can be used instead of their much longer expansions, and this provides for a much more concise and simple notation. The precise way this mechanism works is explained in Section Shortcuts for Constants in RIF's Presentation Syntax.

The precise way in which these directives work is explained in Section Shortcuts for Constants in RIF's Presentation Syntax.

To avoid writing down long IRIs, this document will assume that the following `Prefix` directives have been specified in all the RIF documents under consideration:

- `Prefix(xs http://www.w3.org/2001/XMLSchema#)`. This prefix stands for the XML Schema namespace URI.
- `Prefix(rdf http://www.w3.org/1999/02/22-rdf-syntax-ns#)`. This prefix stands for the RDF URI.
- `Prefix(rif http://www.w3.org/2007/rif#)`. The `rif` prefix stands for the RIF URI.
- `Prefix(func http://www.w3.org/2007/rif-builtin-function#)`. This prefix expands into a URI used for RIF builtin functions.
- `Prefix(pred http://www.w3.org/2007/rif-builtin-predicate#)`. This is the prefix used for RIF builtin predicates.

Using these prefixes and the shorthand mechanism defined in Section Shortcuts for Constants in RIF's Presentation Syntax, we can, for example, abbreviate a constant such as `"http://www.example.org"^^<http://www.w3.org/2007/rif#iri>` into `"http://www.example.org"^^rif:iri`.

1.2.1 Symbol Spaces

Formally, we define symbol spaces as follows.

Definition (Symbol space). A ***symbol space*** is a named subset of the set of all constants, `Const` in RIF. Each symbol in `Const` belongs to exactly one symbol space.

Each symbol space has an associated lexical space, a unique IRI identifying it and a short name. More precisely,

- The ***lexical space*** of a symbol space is a non-empty set of Unicode character strings.
- The ***identifier*** of a symbol space is a sequence of Unicode characters that form an absolute IRI.
- The ***short name*** of a symbol space is an NCName, typically the character sequence after the last '/' or '#' in the symbol space IRI (similar to the XML local name part of a QName).
- Different symbol spaces supported by a dialect cannot share the same identifier or short name.

The identifiers of symbol spaces are **not** themselves constant symbols in RIF.

For convenience we will often use symbol space identifiers to refer to the actual symbol spaces (for instance, we may use "symbol space `xs:string`" instead of "symbol space *identified by* `xs:string`").

RIF dialects are expected to include the following symbol spaces. However, rule sets that are exchanged through RIF can use additional symbol spaces.

- `xs:string` (`http://www.w3.org/2001/XMLSchema#string`, short name: `string`)
- `xs:time` (`http://www.w3.org/2001/XMLSchema#time`, short name: `time`)
- `xs:date` (`http://www.w3.org/2001/XMLSchema#date`), short name: `date`
- `xs:dateTime` (`http://www.w3.org/2001/XMLSchema#dateTime`), short name: `dateTime`)
- `xs:double` (`http://www.w3.org/2001/XMLSchema#double`, short name: `double`)
- `xs:integer` (`http://www.w3.org/2001/XMLSchema#integer`, short name: `integer`)
- `xs:decimal` (`http://www.w3.org/2001/XMLSchema#decimal`, short name: `decimal`)

The lexical spaces of the above symbol spaces are defined in the document [XML-SCHEMA2].

- `xs:dayTimeDuration` (`http://www.w3.org/2001/XMLSchema#dayTimeDuration`, short name: `dayTimeDuration`)
- `xs:yearMonthDuration` (`http://www.w3.org/2001/XMLSchema#yearMonthDuration`, short name: `yearMonthDuration`)

These two symbol spaces represent two subtypes of the XML Schema datatype `xs:duration` with well-defined value spaces, since `xs:duration` does not have a well-defined value space (this may be corrected in later revisions of XML Schema datatypes, in which case the revised datatype would be suitable for RIF DTB). The lexical spaces of the above symbol spaces are defined in the document [XDM].

- `rdf:text` (`http://www.w3.org/1999/02/22-rdf-syntax-ns#text` , short name: `text`).

  This symbol space represents text strings with a language tag attached. The lexical space of `rdf:text` is defined in the document [RDF-TEXT].

- `rdf:XMLLiteral` (`http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral`, short name: `XMLLiteral`).

  This symbol space represents XML content. The lexical space of `rdf:XMLLiteral` is defined in the document [RDF-CONCEPTS].

- `rif:iri` (`http://www.w3.org/2007/rif#iri`, , short name: `iri`, for ***internationalized resource identifiers*** or ***IRI***s).

  Constants in this symbol space are intended to be used in a way similar to RDF resources [RDF-SCHEMA]. The lexical space consists of all absolute IRIs as specified in [RFC-3987]; it is unrelated to the XML primitive type `anyURI`. A `rif:iri` constant must be interpreted as a reference to one and the same object regardless of the context in which that constant occurs.

- `rif:local` (`http://www.w3.org/2007/rif#local`, , short name: `local`, for constant symbols that are not visible outside of the RIF document in which they occur).

  Constants in this symbol space are local to the RIF documents in which they occur. This means that occurrences of the same `rif:local` constant in different documents are viewed as unrelated distinct constants, but occurrences of the same `rif:local` constant in the same document must refer to the same object. The lexical space of `rif:local` is the same as the lexical space of `xs:string`.

Note that, by the associated lexical space, not all unicode strings are syntactically valid lexical parts for all symbol spaces. That is, for instance `"1.2"^^xs:decimal` and `"1"^^xs:integer` are syntactically valid constant because 1.2 and 1 are members of the lexical space of symbol spaces `xs:decimal` and `xs:integer`, respectively. On the other hand, `"a+2"^^xs:decimal` is not a syntactically valid constant, since `a+2` is not part of the lexical space of `xs:decimal`.

We will often refer to constant symbols that come from a particular symbol space, `X`, as `X`-constants. For instance the constants in the symbol space `rif:iri` will be referred to as ***IRI constants*** or `rif:iri` ***constants*** and the constants found in the symbol space `rif:local` as ***local constants*** or `rif:local` ***constants***.

1.2.2 Shortcuts for Constants in RIF's Presentation Syntax

Besides the basic notion

`"literal"^^<identifier>`

RIF's presentation syntax introduces several shortcuts for particular symbol spaces, in order to make the presentation syntax more readable. RIF's presentation syntax for constants is defined by the following EBNF.

```
ANGLEBRACKIRI ::= IRI_REF
SYMSPACE      ::= ANGLEBRACKIRI | CURIE
CURIE         ::= PNAME_LN | PNAME_NS
Const         ::= '"' UNICODESTRING '"^^' SYMSPACE | CONSTSHORT
CONSTSHORT    ::= ANGLEBRACKIRI              // shortcut for "..."^^rif:i
                | CURIE                      // shortcut for "..."^^rif:i
                | '"' UNICODESTRING '"'      // shortcut for "..."^^xs:st
                | NumericLiteral             // shortcut for "..."^^xs:in
                | '_' LocalName              // shortcut for "..."^^rif:l
                | '"' UNICODESTRING '"' '@' languageTag             // sh
```

The EBNF grammar relies on reuse of nonterminals defined in the following grammar productions from other documents:

- `PNAME_LN`, cf. http://www.w3.org/TR/rdf-sparql-query/#rPNAME_LN
- `PNAME_NS`, cf. http://www.w3.org/TR/rdf-sparql-query/#rPNAME_NS
- `languageTag`, cf. http://www.w3.org/2007/OWL/wiki/InternationalizedStringSpec#AbbreviationsGrammar
- `NumericLiteral`, cf. http://www.w3.org/TR/rdf-sparql-query/#rNumericLiteral
- `IRI_REF`, cf. http://www.w3.org/TR/rdf-sparql-query/#rIRI_REF
- `LocalName`, cf. http://www.w3.org/TR/2006/REC-xml-names11-20060816/#NT-LocalPart
- `UNICODESTRING`, any Unicode string where quotes are escaped and additionally all the other escape sequences defined in http://www.w3.org/

TR/rdf-sparql-query/#grammarEscapes and http://www.w3.org/TR/rdf-sparql-query/#codepointEscape.

In this grammar, `CURIE` stands for *compact IRIs* [CURIE], which are used to abbreviate symbol space IRIs. For instance, one can write `"http://www.example.org"^^rif:iri` instead of `"http://www.example.org"^^<http://www.w3.org/2007/rif#iri>`, where `rif` is a prefix defined in Section Base and Prefix Directives. <p>Apart from compact IRIs, there exist convenient shortcut notations for constants in specific symbol spaces, namely for constants in the symbol spaces `rif:iri`, `xs:string`, `xs:integer`, `xs:decimal`, `xs:double`, and `rif:local`:

- Constants in the the symbol space `rif:iri` can be abbreviated in two ways, either by simply using an absolute or relative IRI enclosed in angle brackets, or by writing a compact IRI. The symbol space identifier is dropped in both of these alternatives. For instance `<http://www.example.org/xyz>` is a valid abbreviation for `"http://www.example.org/xyz"^^rif:iri` and , `ex:xyz` is a valid abbreviation for this constant, if the directive

  ```
  Prefix(ex http://www.example.org/)
  ```

  is present in the RIF document in question.

- Constants in the symbol space `xs:string` can be abbreviated by simply using quoted strings, i.e. `"My String!"` is a valid abbreviation for the constant `"My String!"^^xs:string` (which in turn is itself an abbreviation for `"My String!"^^<http://www.w3.org/2001/XMLSchema#string>`).
- Numeric constants can be abbreviated using the grammar rules for NumericLiterals from the [SPARQL] grammar: Integers can be written directly (without quotation marks and explicit symbol space identifier) and are interpreted as constants in the symbol space `xs:integer`; decimal numbers for which there is '.' in the number but no exponent are interpreted as constants in the symbol space `xs:decimal`; and numbers with exponents are interpreted as `xs:double`. For instance, one could use `1.2` and `1` as shortcuts for `"1.2"^^xs:decimal` and `"1"^^xs:integer`, respectively. However, there is no shortcut for `"1"^^xs:decimal`.
- The shortcut notation for `rif:local` applies to only a subset of the lexical space of syntactically valid lexical parts of constants in this symbol space: We allow "_"-prefixed unicode strings which are also valid XML NCNames as defined in [XML-NS]. For other constants in the `rif:local` symbol space one has to use the long notation. That is, for instance `_myLocalConstant` is a valid abbreviation for the constant `"myLocalConstant"^^rif:local`, whereas `"http://www.example.org"^^rif:local` cannot be abbreviated.

1.2.3 Relative IRIs

Relative IRIs in RIF documents are resolved with respect to the ***base IRI***. Relative IRIs are combined with base IRIs as per Uniform Resource Identifier (URI): Generic Syntax [RFC-3986] using only the basic algorithm in Section 5.2. Neither Syntax-Based Normalization nor Scheme-Based Normalization (described in sections 6.2.2 and 6.2.3 of RFC-3986) are performed. Characters additionally allowed in IRI references are treated in the same way that unreserved characters are treated in URI references, per section 6.5 of Internationalized Resource Identifiers (IRIs) [RFC-3987].

Base IRIs are specified using the `Base` directive described in Section Base and Prefix Directives. At most one base directive per document is allowed. In the XML syntax, base IRIs are specified using the attribute `xml:base`.

For instance, the constant <./xyz> or "./xyz"^^rif:iri are both valid abbreviations in RIF for the constant `http://www.example.org/xyz"^^rif:iri`, if the following directive is present in the document:

```
Base(http://www.example.org)
```

1.3 Primitive Datatypes

Datatypes in RIF are symbol spaces which have special semantics. That is, each datatype is characterized by a fixed lexical space, value space and lexical-to-value-mapping.

Definition (Primitive datatype). A ***primitive datatype*** (or just a *datatype*, for short) is a symbol space that has

- an associated set, called the ***value space***, and
- a mapping from the lexical space of the symbol space to the value space, called ***lexical-to-value-space mapping***. □

Semantic structures are always defined with respect to a particular set of datatypes, denoted by ***DTS***. In a concrete dialect, ***DTS*** always includes the datatypes supported by that dialect. RIF dialects are expected to support the following primitive datatypes. However, RIF dialects may include additional datatypes.

- `xs:string`
- `xs:time`
- `xs:date`
- `xs:dateTime`
- `xs:double`
- `xs:integer`
- `xs:decimal`

- `xs:dayTimeDuration`
- `xs:yearMonthDuration`
- `rdf:text`
- `rdf:XMLLiteral`

Their value spaces and the lexical-to-value-space mappings are defined as follows:

- For the XML Schema datatypes of RIF, namely `xs:double`, `xs:integer`, `xs:decimal`, `xs:time`, `xs:dateTime`, and `xs:string` the value spaces and the lexical-to-value-space mappings are defined in the XML Schema specification [XML-SCHEMA2].
- The value spaces and the lexical-to-value-space mappings for the primitive datatypes `xs:dayTimeDuration` and `xs:yearMonthDuration` are defined in the XQuery 1.0 and XPath 2.0 Data Model [XDM].
- The value space and the lexical-to-value-space mapping for `rdf:text` are defined in the document [RDF-TEXT].
- The value space and lexical-to-value-space mapping for the primitive datatype `rdf:XMLLiteral` is defined in RDF [RDF-CONCEPTS].

Note that the value space and the lexical-to-value-space mapping for `rdf:text` defined here are compatible with RDF's semantics for string literals with named tags [RDF-SEMANTICS]. Moreover, the value space and the lexical-to-value-space mapping for `xs:string` are compatible with RDF's semantics for plain literals. RIF implementations MAY choose to interpret `xs:string` and its subtypes as subtypes of `rdf:text` following Section 3.1 of [RDF-TEXT], i.e., interpreting strings as texts with an empty language tag.

Editor's Note: Whether or not we allow the treatment of `xs:string` as a subtype of `rdf:text` in RIF implementations is still under discussion, cf. the mail thread starting at http://lists.w3.org/Archives/Public/public-rif-wg/2008Nov/0067.html.

Editor's Note: Some clarification is needed with respect to the value space of `rdf:XMLLiteral` which will hopefully be resolved by an erratum to the RDF spec., cf. the mail thread starting at http://lists.w3.org/Archives/Public/public-rif-wg/2008Dec/0013.html.

2 Syntax and Semantics of Built-ins

2.1 Syntax of Built-ins

A RIF built-in function or predicate is a special case of externally defined terms, which are defined in RIF Framework for Logic Dialects and also reproduced in the direct definition of RIF Basic Logic Dialect (RIF-BLD).

In RIF's presentation syntax built-in predicates and functions are syntactically represented as external terms of the form:

```
'External' '(' Expr ')'
```

where `Expr` is a positional term as defined in RIF Framework for Logic Dialects (see also in RIF Basic Logic Dialect). For RIF's normative syntax, see the XML Serialization Framework in RIF-FLD, or, specifically for RIF-BLD, see XML Serialization Syntax for RIF-BLD.

RIF-FLD introduces the notion of an external schema to describe both both the syntax and semantics of exernally defined terms. In the special case of a RIF built-in, external schemas have an especially simple form. A built-in named `f` that takes `n` arguments has the schema

$(?X_1 \ldots ?X_n ; \quad f(?X_1 \ldots ?X_n))$

Here $f(?X_1 \ldots ?X_n)$ is the actual term that is used to refer to the built-in (in expressions of the form $External(f(?X_1 \ldots ?X_n))$) and $?X_1 \ldots ?X_n$ is the list of all variables in that term.

For convenience, a complete definition of external schemas is reproduced in Appendix: Schemas for Externally Defined Terms.

2.2 Semantics of Built-ins

The semantics of external terms in RIF-FLD and RIF-BLD is defined using two mappings: $I_{external}$ and $I_{truth} \circ I_{external}$.

- $I_{external}$. This mapping takes an external schema, σ, and returns a mapping, $I_{external}(\sigma)$.

  If σ represents a built-in function, $I_{external}(\sigma)$ must be that function.

  For each built-in function with external schema σ, the present document specifies the mapping $I_{external}(\sigma)$.

- I_{truth}. This mapping takes an element of the domain of interpretation and returns a truth value.

  In RIF logical semantics, this mapping is used to assign truth values to formulas. In the special case of RIF built-ins, it is used to assign truth values to RIF built-in predicates. The built-in predicates can have the truth values **t** or **f** only.

For a built-in predicate with schema σ, RIF-FLD and RIF-BLD require that the truth-valued mapping $I_{truth} \circ I_{external}(\sigma)$ must agree with the specification of the corresponding built-in predicate.

For each RIF built-in predicate with schema σ, the present document specifies $I_{truth} \circ I_{external}(\sigma)$.

3 List of RIF Built-in Predicates and Functions

This section provides a catalogue defining the syntax and semantics of a list of built-in predicates and functions in RIF. For each built-in, the following is defined:

1. The ***name*** of the built-in.
2. The ***external schema*** of the built-in.
3. For a built-in function, how it maps its arguments into a result.

   As explained in Section Semantics of Built-ins, this corresponds to the mapping $I_{external}(\sigma)$ in the formal semantics of RIF-FLD and RIF-BLD, where σ is the external schema of the built-in.

4. For a built-in predicate, its truth value when the arguments are substituted with values in the domain.

   As explained in Section Semantics of Built-ins, this corresponds to the mapping $I_{truth} \circ I_{external}(\sigma)$ in the formal semantics of RIF-FLD and RIF-BLD, where σ is the external schema of the built-in.

5. The ***domains*** for the arguments of the built-in.

   Typically, built-in functions and predicates are defined over the value spaces of appropriate datatypes, i.e. the domains of the arguments. When an argument falls outside of its domain, it is understood as an error. Since this document defines a model-theoretic semantics for RIF built-ins, which does not support the notion of an error, the definitions leave the values of the built-in predicates and functions **unspecified** in such cases. This means that if one or more of the arguments is not in its domain, the value of $I_{external}(\sigma)(a_1 \ldots a_n)$ is unspecified. In particular, this means it can vary from one implementation to another. Similarly, $I_{truth} \circ I_{external}(\sigma)(a_1 \ldots a_n)$ is unspecified when an argument is not in its domain.

   This indeterminacy in case of an error implies that applications should not make any assumptions about the values of built-ins in such situations. Implementations are even allowed to abort in such cases and the only safe way to communicate rule sets that contain built-ins among RIF-compliant systems is to use datatype guards.

Many built-in functions and predicates described below are adapted from [XPath-Functions] and, when appropriate, we will refer to the definitions in that specification in order to avoid copying them.

3.1 Guard Predicates for Datatypes

RIF defines guard predicates for all datatypes in Section Primitive Datatypes.

- *Schema*: The schemas for these predicates have the general form

  `( ?arg1; pred:is`*DATATYPE*` ( ?arg1 ) )`

  Here, *DATATYPE* is the short name for a datatype. For instance, we use `pred:isString` for the guard predicate for `xs:string`, `pred:isText` for the guard predicate for `rif:text`, or `pred:isXMLLiteral` for the guard predicate for`rdf:XMLLiteral`. Parties defining their own datatypes to be used in RIF exchanged rules may define their own guard predicates for these datatypes. Labels used for such additional guard predicates for datatypes not mentioned in the present document MAY follow a similar naming convention where applicable without creating ambiguities with predicate names defined in the present document. Particularly, upcoming W3C specifications MAY - but 3rd party dialects MUST NOT - reuse the `pred:` namespace for such guard predicates.

  Editor's Note: The formulation of the clause on namespace-reuse is under discussion, for instance, whether we shall allow guards and negative guards for all of the XML Schema primitive datatypes under the `pred:` namespace.

- *Domain*:

  Guard predicates do not depend on a specific domain.

- *Mapping:*

  $I_{truth} \circ I_{external}$(?arg$_1$; pred:is*DATATYPE* (?arg$_1$))(s$_1$) = **t** if and only if s$_1$ is in the value space of *DATATYPE* and **f** otherwise.

Accordingly, the following schemas are defined.

3.1.1 `pred:isInteger`

- *Schema*:

  `( ?arg1; pred:isInteger( ?arg1 ) )`

3.1.2 pred:isDecimal

- *Schema*:

  (?arg$_1$; pred:isDecimal (?arg$_1$))

3.1.3 pred:isDouble

- *Schema*:

  (?arg$_1$; pred:isDouble (?arg$_1$))

3.1.4 pred:isString

- *Schema*:

  (?arg$_1$; pred:isString (?arg$_1$))

3.1.5 pred:isTime

- *Schema*:

  (?arg$_1$; pred:isTime (?arg$_1$))

3.1.6 pred:isDate

- *Schema*:

  (?arg$_1$; pred:isDate (?arg$_1$))

3.1.7 pred:isDateTime

- *Schema*:

  (?arg$_1$; pred:isDateTime (?arg$_1$))

3.1.8 pred:isDayTimeDuration

- *Schema*:

  (?arg$_1$; pred:isDayTimeDuration (?arg$_1$))

3.1.9 `pred:isYearMonthDuration`

- *Schema*:

  ```
  ( ?arg1; pred:isYearMonthDuration ( ?arg1 ) )
  ```

3.1.10 `pred:isXMLLiteral`

- *Schema*:

  ```
  ( ?arg1; pred:isXMLLiteral ( ?arg1 ) )
  ```

3.1.11 `pred:isText`

- *Schema*:

  ```
  ( ?arg1; pred:isText ( ?arg1 ) )
  ```

Future dialects may extend this list of guards to other datatypes, but RIF does not require guards for all datatypes.

3.2 Negative Guard Predicates for Datatypes

Likewise, RIF defines negative guard predicates for all datatypes in Section Primitive Datatypes.

- *Schema*: The schemas for negative guards have the general form

  ```
  ( ?arg1; pred:isNotDATATYPE ( ?arg1 ) )
  ```

  Here, *DATATYPE* is the short name for one of the datatypes mentioned in this document. For instance, we use `pred:isNotString` for the negative guard predicate for `xs:string`, `pred:isNotText` for the negative guard predicate for `rif:text`, or `pred:isNotXMLLiteral` for the negative guard predicate for `rdf:XMLLiteral`. Parties defining their own datatypes to be used in RIF exchanged rules may define their own negative guard predicates for these datatypes. Labels used for such additional negative guard predicates for datatypes not mentioned in the present document MAY follow a similar naming convention where applicable without creating ambiguities with predicate names defined in the present document. Particularly, upcoming W3C specifications MAY, but 3rd party dialects MUST NOT reuse, the `pred:` namespace for such negative guard predicates.

Editor's Note: The formulation of the clause on namespace-reuse is under discussion, for instance, whether we shall allow guards and negative guards for all of the XML Schema primitive datatypes under the `pred:` namespace.

- *Domain*:

  Negative guard predicates do not depend on a specific domain.

- *Mapping:*

  $I_{truth} \circ I_{external}$(?arg$_1$; pred:isNot*DATATYPE* (?arg$_1$))(s$_1$) = **f** if and only if s$_1$ is in the value space of *DATATYPE* and **t** otherwise.

Accordingly, the following schemas are defined.

3.2.1 `pred:isNotInteger`

- *Schema*:

  ```
  ( ?arg1; pred:isNotInteger ( ?arg1 ) )
  ```

3.2.2 `pred:isNotDecimal`

- *Schema*:

  ```
  ( ?arg1; pred:isNotDecimal ( ?arg1 ) )
  ```

3.2.3 `pred:isNotDouble`

- *Schema*:

  ```
  ( ?arg1; pred:isNotDouble ( ?arg1 ) )
  ```

3.2.4 `pred:isNotString`

- *Schema*:

  ```
  ( ?arg1; pred:isNotString ( ?arg1 ) )
  ```

3.2.5 `pred:isNotTime`

- *Schema*:

(?arg$_1$; pred:isNotTime (?arg$_1$))

3.2.6 pred:isNotDate

- *Schema*:

  (?arg$_1$; pred:isNotDate (?arg$_1$))

3.2.7 pred:isNotDateTime

- *Schema*:

  (?arg$_1$; pred:isNotDateTime (?arg$_1$))

3.2.8 pred:isNotDayTimeDuration

- *Schema*:

  (?arg$_1$; pred:isNotDayTimeDuration (?arg$_1$))

3.2.9 pred:isNotYearMonthDuration

- *Schema*:

  (?arg$_1$; pred:isNotYearMonthDuration (?arg$_1$))

3.2.10 pred:isNotXMLLiteral

- *Schema*:

  (?arg$_1$; pred:isNotXMLLiteral (?arg$_1$))

3.2.11 pred:isNotText

- *Schema*:

  (?arg$_1$; pred:isNotText (?arg$_1$))

Future dialects may extend this list of guards to other datatypes, but RIF does not require negative guards for all datatypes.

3.3 Datatype Conversion and Datatypes Checking

In the following, we adapt several cast functions according to the conversions defined in Section 17.1 of [XPath-Functions]. Note that some of these conversions are only partially defined, which affects the domains of these cast functions.

Editor's Note: Due to the subtle differences in casting, e.g., concerning error handling, between RIF and [XPath-Functions], the definitions of cast functions might still need refinement in terms of defining the domains in future versions of this draft. Also, the definition of the mappings need refinement. See http://lists.w3.org/Archives/Public/public-rif-wg/2008Nov/0067.html, esp. the response to item 6)

Likewise we define a conversion predicate useful for converting between `rif:iri` constants and strings, as well as a predicate to check the datatype of a constant.

3.3.1 `xs:double`

- *Schema*:

  `( ?arg1 xs:double ( ?arg1 ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:double` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg$_1$; xs:double (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:double` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.2 `xs:integer`

- *Schema*:

  `( ?arg1 xs:integer ( ?arg1 ) )`

- *Domain*:

The union of the (subsets of the) value spaces of datatypes castable to `xs:integer` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg$_1$; xs:integer (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:integer` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.3 `xs:decimal`

- *Schema*:

  `( ?arg`$_1$`; xs:decimal ( ?arg`$_1$` ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:decimal` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg$_1$; xs:decimal (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:decimal` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.4 `xs:time`

- *Schema*:

  `( ?arg`$_1$`; xs:time ( ?arg`$_1$` ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:time` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

$I_{external}$(?arg$_1$; xs:time (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:time` according to Section 17.1 of [XPath-Functions].

If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.5 `xs:date`

- *Schema*:

  `( ?arg`$_1$`; xs:date ( ?arg`$_1$ ` ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:date` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg$_1$; xs:date (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:date` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.6 `xs:dateTime`

- *Schema*:

  `( ?arg`$_1$`; xs:dateTime ( ?arg`$_1$ ` ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:dateTime` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg$_1$; xs:dateTime (?arg$_1$))(s$_1$) = s$_1$' such that s$_1$' is the conversion of s$_1$ to the value space of `xs:dateTime` according to Section 17.1 of [XPath-Functions].

If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.7 `xs:dayTimeDuration`

- *Schema*:

  `( ?arg1 xs:dayTimeDuration ( ?arg1 ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:dayTimeDuration` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg1; xs:dayTimeDuration (?arg1))(s_1) = s_1' such that s_1' is the conversion of s_1 to the value space of `xs:dayTimeDuration` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.8 `xs:yearMonthDuration`

- *Schema*:

  `( ?arg1 xs:yearMonthDuration ( ?arg1 ) )`

- *Domain*:

  The union of the (subsets of the) value spaces of datatypes castable to `xs:yearMonthDuration` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg1; xs:yearMonthDuration (?arg1))(s_1) = s_1' such that s_1' is the conversion of s_1 to the value space of `xs:yearMonthDuration` according to Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain or outside the partial conversions defined in [XPath-Functions], the value of the function is left unspecified.

3.3.9 `xs:string`

- *Schema*:

  `( ?arg1; xs:string ( ?arg1 ) )`

- *Domain*:

  The union of the value space of `rdf:XMLLiteral` with the value spaces of datatypes castable to `xs:string` according to Section 17.1 of [XPath-Functions].

- *Mapping:*

  $I_{external}$(?arg1; xs:string(?arg1))(s_1) = s_1' such that

  - s_1' is the conversion of s_1 to the value space of `xs:string` according to the table in Section 17.1 of [XPath-Functions], in case s_1 is in the value space of a datatype mentioned there.
  - s_1' is the string in the lexical space of `rdf:XMLLiteral` correspoding to s_1 (cf. [RDF-CONCEPTS]) ,in case s_1 is in the value space of `rdf:XMLLiteral`.

  If the argument is outside of its domain, the value of the function is left unspecified.

Note: Since RIF implementations MAY choose to interpret `xs:string` and its subtypes as subtypes of `rdf:text` following Section 3.1 of [RDF-TEXT], in such implementations this cast function also serves for conversions to `rdf:text`.

Editor's Note: Whether or not we allow the treatment of `xs:string` as a subtype of `rdf:text` in RIF implementations is still under discussion, cf. the mail thread starting at http://lists.w3.org/Archives/Public/public-rif-wg/2008Nov/0067.html.

Editor's Note: Casting from rdf:XMLLiteral to xs:string is still under discussion.

3.3.10 `rdf:XMLLiteral`

- *Schema*:

`( ?arg1; rdf:XMLLiteral ( ?arg1 ) )`

- *Domain*:

  The intersection of the value space of `xs:string` with the lexical space of `rdf:XMLLiteral`, i.e. an `xs:string` can be cast to `rdf:XMLLiteral` if and only if its value is in the lexical space of `rdf:XMLLiteral` as defined in Resource Description Framework (RDF): Concepts and Abstract Syntax

- *Mapping:*

  $I_{external}$(?arg1; xs:XMLLiteral (?arg1))(s_1) = s_1' such that s_1' is the XMLLiteral corresponding to the given string s_1.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.3.11 `pred:iri-string`

Conversions from `rif:iri` to `xs:string` and vice versa cannot be defined by the casting functions as above since `rif:iri` is not a datatype with a well-defined value space.

To this end, since conversions from IRIs (resources) to strings are a needed feature for instance for conversions between RDF formats (see example below), we introduce a built-in predicate which supports such conversions.

- *Schema*:

  `( ?arg1 ?arg2; pred:iri-string ( ?arg1, ?arg2 ) )`

- *Domains*:

  The first argument is not restricted by a specific domain, the second argument is the value space of `xs:string`.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; pred:iri-string (?arg1 ?arg2))(iri_1 str_1) = **t** if and only if str_1 is a string in the lexical space of `rif:iri` and iri_1 is an element of the domain such that I("str_1"^^`rif:iri`) = iri_1 holds in the current interpretation.

  Note that this definition restricts allowed RIF interpretations in such a way that the interpretation of `pred:iri-string` always needs to comply with respect to the symbols in the `rif:iri` symbol space for the first

argument and elements of the `xs:string` value space for the second argument. The truth value of the predicate is left unspecified for other elements of the domain.

This predicate could be usable for instance to map telephone numbers between an RDF Format for vCard (http://www.w3.org/TR/vcard-rdf) and FOAF (http://xmlns.com/foaf/0.1/). vCard stores telephone numbers as string literals, whereas FOAF uses resources, i.e., URIs with the tel: URI-scheme. So, a mapping from FOAF to vCard would need to convert the `tel:` URI to a string and then cut off the first four characters ("tel:"). Such a mapping expressed in RIF could involve e.g. a rule as follows:

```
...
Prefix( VCard http://www.w3.org/TR/vcard-rdf#)
Prefix( foaf http://xmlns.com/foaf/0.1/)
...
Forall ?X ?foafTelIri ?foafTelString (
  ?X[ VCard:tel -> External( func:substring( ?foafTelString 4 ) ]
        And ( ?X[ foaf:phone -> ?foafTelIri ]
              External( pred:iri-string( ?foafTelIri ?foafTelStrin
```

3.3.12 `pred:hasDatatype`

Extractions of the Datatype from a constant cannot be defined by a function (like for instance in SPARQL's `datatype` function) since the value spaces of datatypes may overlap.

To this end, we introduce a built-in predicate which supports extraction of the datatypes for a constant at hand.

- *Schema*:

  `( ?arg1 ?arg2; pred:hasDatatype ( ?arg1, ?arg2 ) )`

- *Domains*:

  None of the arguments is restricted to a specific domain.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; pred:hasDatatype (?arg1 ?arg2))(const1 iri1) = **t** if and only if in the current interpretation iri1 = I("*DatatypeIRI*"`^^rif:iri`) where *DatatypeIRI* is the IRI identifier of a datatype *d* and const1 is in the value space of *d*.

Editor's Note: It is still under discussion in the WG whether this predicate should restrict the domain of the second argument rather to strings that represent valid

IRIs than just being true for any constants that have the same **interpretation** as the particular rif:iri representing the datatype.

This predicate can be usable for extracting the datatype from a constant but due to the overlap of the calue spaces of datatypes, such extraction is not necessarily unique; for example, the following is entailed in any RIF ruleset:

```
And ( External( pred:hasDatatype( "1.0"^^xs:decimal xs:decimal ) )
      External( pred:hasDatatype( "1.0"^^xs:decimal xs:integer ) )
      External( pred:hasDatatype( "1.0"^^xs:decimal xs:double  ) ) )
```

Editor's Note: Note that this example shows that pred:hasDatatype is not adequate for emulating SPARQL's datatype function http://www.w3.org/TR/rdf-sparql-query/#func-datatype, cf. the mail thread starting at http://lists.w3.org/Archives/Public/public-rif-wg/2008Nov/0067.html

The following example shows that also whether or not a RIF implementation that treats `xs:string` as a subtype of `rdf:text` may affect the entailments for `pred:hasDatatype`:

```
Forall ?X (
  ?P [ ex:nameType -> ?D ] :-
        And ( ?P[ foaf:name -> ?N ]
              External( pred:hasDatatype( ?N ?D ) ) ) )
ex:alice [foaf:name -> "Alice"]
```

In a RIF implementation that treats `xs:string` as a subtype of `rdf:text`, following Section 3.1 of [RDF-TEXT], this ruleset would entail both `ex:alice [ex:nameType -> rdf:text]` and `ex:alice [ex:nameType -> xs:string]`.

Editor's Note: Whether or not we allow the treatment of `xs:string` as a subtype of `rdf:text` in RIF implementations is still under discussion, cf. the mail thread starting at http://lists.w3.org/Archives/Public/public-rif-wg/2008Nov/0067.html.

Editor's Note: It is still under discussion in the WG whether an additional predicate `pred:hasNotDatatype` should be added, cf. ISSUE-80.

3.4 Numeric Functions and Predicates

The following functions and predicates are adapted from the respective numeric functions and operators in [XPath-Functions].

3.4.1 Numeric Functions

The following numeric built-in functions `func:numeric-add`, `func:numeric-subtract`, `func:numeric-multiply`, `func:numeric-divide`, `func:numeric-integer-divide`, and `func:numeric-mod` are defined in accordance with their corresponding operators in [XPath-Functions].

3.4.1.1 `func:numeric-add` (adapted from `op:numeric-add`)

- *Schema*:

  `(?arg1 ?arg2; func:numeric-add(?arg1 ?arg2))`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for both arguments.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-add(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-add(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.4.1.2 `func:numeric-subtract` (adapted from `op:numeric-subtract`)

- *Schema*:

  `(?arg1 ?arg2; func:numeric-subtract( ?arg1 ?arg2) )`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for both arguments.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-subtract(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-subtract(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.4.1.3 `func:numeric-multiply` (adapted from `op:numeric-multiply`)

- *Schema*:

  `(?arg1 ?arg2; func:numeric-multiply( ?arg1 ?arg2) )`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for both arguments.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-multiply(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-multiply(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.4.1.4 `func:numeric-divide` (adapted from `op:numeric-divide`)

- *Schema*:

  `(?arg1 ?arg2; func:numeric-divide( ?arg1 ?arg2) )`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for the first argument and `xs:integer`, `xs:double`, or `xs:decimal` without zero for the second argument.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-divide(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-divide(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified, which here particularly means that RIF does not prescribe the behavior on division by zero.

3.4.1.5 `func:numeric-integer-divide` (adapted from `op:numeric-integer-divide`)

- *Schema*:

`(?arg1 ?arg2; func:numeric-integer-divide( ?arg1 ?arg2) )`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for the first argument and `xs:integer`, `xs:double`, or `xs:decimal` without zero for the second argument.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-integer-divide(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-integer-divide(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified, which here particularly means that RIF does not prescribe the behavior on division by zero.

3.4.1.6 `func:numeric-mod` (adapted from `op:numeric-mod`)

- *Schema*:

  `(?arg1 ?arg2; func:numeric-mod( ?arg1 ?arg2) )`

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for the first argument and `xs:integer`, `xs:double`, or `xs:decimal` without zero for the second argument.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2; func:numeric-mod(?arg1 ?arg2))(a_1 a_2) = *res* such that *res* is the result of op:numeric-mod(a_1, a_2) as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified, which here particularly means that RIF does not prescribe the behavior if the second argument is zero.

3.4.2 Numeric Predicates

3.4.2.1 `pred:numeric-equal` (adapted from `op:numeric-equal`)

- *Schema*:

(?arg_1 ?arg_2; pred:numeric-equal(?arg_1 ?arg_2))

- *Domains*:

  The value spaces of `xs:integer`, `xs:double`, or `xs:decimal` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}$(?$arg_1$?arg_2; pred:numeric-equal(?arg_1 ?arg_2))(a_1 a_2) = **t** if and only if op:numeric-equal(a_1, a_2) returns `true`, as defined in [XPath-Functions].

  If an argument value is outside of its domain, the truth value of the function is left unspecified.

The following numeric built-in predicates `pred:numeric-less-than` and `pred:numeric-greater-than` are defined analogously with respect to their corresponding operators in [XPath-Functions].

3.4.2.2 `pred:numeric-less-than` (adapted from `op:numeric-less-than`)

- *Schema*:

  (?arg_1 ?arg_2; pred:numeric-less-than(?arg_1 ?arg_2))

3.4.2.3 `pred:numeric-greater-than` (adapted from `op:numeric-greater-than`)

- *Schema*:

  (?arg_1 ?arg_2; pred:numeric-greater-than(?arg_1 ?arg_2))

3.4.2.4 `pred:numeric-not-equal`

- *Schema*:

  (?arg_1 ?arg_2; pred:numeric-not-equal(?arg_1 ?arg_2))

The predicate `pred:numeric-not-equal` has the same domains as `pred:numeric-equal` and is true whenever `pred:numeric-equal` is false.

3.4.2.5 `pred:numeric-less-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:numeric-less-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:numeric-less-than-or-equal` has the same domains as `pred:numeric-equal` and is true whenever `pred:numeric-equal` is true or `pred:numeric-less-than` is true.

3.4.2.6 `pred:numeric-greater-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:numeric-greater-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:numeric-greater-than-or-equal` has the same domains as `pred:numeric-equal` and is true whenever `pred:numeric-equal` is true or `pred:numeric-greater-than` is true.

3.5 Functions and Predicates on Strings

The following functions and predicates are adapted from the respective functions and operators on strings in [XPath-Functions].

Editor's Note: The following treatment of built-ins which may have multiple arities is a strawman proposal currently under discussion in the working group.

In the following, we encounter several versions of some built-ins with varying arity, since XPath and XQuery allow overloading, i.e. the same function or operator name occurring with different arities. We treat this likewise in RIF, by numbering the different versions of the respective built-ins and treating the unnumbered version as syntactic sugar, i.e. for instance instead of `External( func:concat2( str1, str2) )` and `External( func:concat3( str1 str2 str3 ) )` we allow the equivalent forms `External( func:concat( str1, str2) )` and `External( func:concat( str1 str2 str3 ) )`. Note that this is really purely syntactic sugar, and does not mean that for external predicates and functions we lift the restriction made in BLD that each function and predicate has a unique assigned arity. Those schemata for which we allow this syntactic sugar, appear in the same box.

3.5.1 Functions on Strings

3.5.1.1 `func:compare` (adapted from `fn:compare`)

- *Schema*:

```
( ?comparand1 ?comparand2;
func:compare1(?comparand1 ?comparand2) )

( ?comparand1 ?comparand2 ?collation;
func:compare2(?comparand1 ?comparand2 ?collation) )
```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping:*

  $I_{external}$((?comparand1 ?comparand2; func:compare1(?comparand1 ?comparand2))(s_1 s_2) = *res* such that *res* = `-1`, `0`, or `1` (from the value space of `xs:integer`), depending on whether the value of the s_1 is respectively less than, equal to, or greater than the value of s_2, according to the rules of the collation that is used. I.e., this function computes the result of fn:compare(s_1, s_2) as defined in [XPath-Functions], in case all arguments belong to their domains.

  If an argument value is outside of its domain, the value of the function is left unspecified.

The following schemata are defined analogously with respect to their corresponding functions and operators as defined in [XPath-Functions] and we only give informal descriptions of the respective mappings $I_{external}$.

3.5.1.2 `func:concat` (adapted from `fn:concat`)

- *Schema*:

```
( ?arg1; func:concat1( ?arg1 ) )

( ?arg1 ?arg2; func:concat2(?arg1 ?arg2 ) )

...

( ?arg1  ?arg2 ... ?argn; func:concatn(?arg1 ?arg2
... ?argn ) )
```

- *Domains*:

  Following the definition of fn:concat this function accepts xs:anyAtomicType arguments and casts them to `xs:string`. Thus, the domain for all arguments is the union of all value spaces castable to String

`xs:string` as defined in Section Cast Functions and Conversion Predicates for Datatypes and rif:iri above.

- *Mapping (informal):*

  Returns the `xs:string` that is the concatenation of the values of its arguments after conversion.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.3 `func:string-join` (adapted from `fn:string-join`)

- *Schema*:

  ```
  ( ?arg1 ?arg2; func:string-join2(?arg1 ?arg2 ) )

  ( ?arg1 ?arg2 ?arg3; func:string-
  join3(?arg1 ?arg2 ?arg3 ) )

  ...

  ( ?arg1  ?arg2 ... ?argn; func:string-joinn(?arg1 ?arg2
  ... ?argn ) )
  ```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns a xs:string created by concatenating the arguments 1 to (*n*-1) using the n^{th} argument as a separator.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.4 `func:substring` (adapted from `fn:substring`)

- *Schema*:

  ```
  ( ?sourceString ?startingLoc;
  func:substring1( ?sourceString ?startingLoc) )

  ( ?sourceString ?startingLoc ?length ;
  func:substring2( ?sourceString ?startingLoc ?length) )
  ```

- *Domains*:

The value space of `xs:string` for ?sourceString and the union of the value spaces of `xs:integer`, `xs:long`, or `xs:decimal` for the remaining two arguments.

- *Mapping (informal):*

  Returns the portion of the value of ?sourceString beginning at the position indicated by the value of ?startingLoc and continuing for the number of characters indicated by the value of ?length. The characters returned do not extend beyond ?sourceString. If ?startingLoc is zero or negative, only those characters in positions greater than zero are returned.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.5 `func:string-length` (adapted from `fn:string-length`)

- *Schema*:

  ```
  ( func:string-length1() )

  ( ?arg ; func:string-length2( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:string` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` equal to the length in characters of the argument if it is a `xs:string`, returns 0 when called without an argument.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.6 `func:upper-case` (adapted from `fn:upper-case`)

- *Schema*:

  ```
  ( ?arg ; func:upper-case( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:string` for ?arg.

- *Mapping (informal):*

Returns the value of ?arg after translating every character to its upper-case correspondent as defined in the appropriate case mappings section in the Unicode standard.

If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.7 `func:lower-case` (adapted from `fn:lower-case`)

- *Schema*:

```
( ?arg ; func:lower-case( ?arg ) )
```

- *Domain*:

  The value space of `xs:string` for ?arg.

- *Mapping (informal):*

  Returns the value of ?arg after translating every character to its lower-case correspondent as defined in the appropriate case mappings section in the Unicode standard.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.8 `func:encode-for-uri` (adapted from `fn:encode-for-uri`)

- *Schema*:

```
( ?arg ; func:encode-for-uri( ?arg ) )
```

- *Domain*:

  The value space of `xs:string` for ?arg.

- *Mapping (informal):*

  This function encodes reserved characters in an `xs:string` that is intended to be used in the path segment of a URI. It is invertible but not idempotent.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.9 `func:iri-to-uri` (adapted from `fn:iri-to-uri`)

- *Schema*:

```
( ?iri ; func:iri-to-uri ( ?iri ) )
```

- *Domain*:

  The value space of `xs:string` for ?iri.

- *Mapping (informal):*

  This function converts an `xs:string` containing an IRI into a URI according to the rules spelled out in Section 3.1 of RFC 3987. It is idempotent but not invertible.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.10 `func:escape-html-uri` (adapted from `fn:escape-html-uri`)

- *Schema*:

  ```
  ( ?uri ;func:escape-html-uri( ?uri ) )
  ```

- *Domain*:

  The value space of `xs:string` for ?uri.

- *Mapping (informal):*

  This function escapes all characters except printable characters of the US-ASCII coded character set, specifically the octets ranging from 32 to 126 (decimal). The effect of the function is to escape a URI in the manner html user agents handle attribute values that expect URIs. Each character in $uri to be escaped is replaced by an escape sequence, which is formed by encoding the character as a sequence of octets in UTF-8, and then representing each of these octets in the form %HH, where HH is the hexadecimal representation of the octet. This function must always generate hexadecimal values using the upper-case letters A-F.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.11 `func:substring-before` (adapted from `fn:substring-before`)

- *Schema*:

  ```
  ( ?arg1 ?arg2; func:substring-before1( ?arg1 ?arg2 ) )
  ```

```
( ?arg1 ?arg2 ?collation; func:substring-
before2( ?arg1 ?arg2 ?collation ) )
```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns the substring of the value of ?arg_1 that precedes in the value of ?arg_1 the first occurrence of a sequence of collation units that provides a minimal match to the collation units of ?arg_2 according to the collation that is used.

  If any argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.12 `func:substring-after` (adapted from `fn:substring-after`)

- *Schema*:

```
( ?arg1 ?arg2; func:substring-after1( ?arg1 ?arg2 ) )

( ?arg1 ?arg2 ?collation; func:substring-
after2( ?arg1 ?arg2 ?collation ) )
```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns the substring of the value of ?arg_1 that follows in the value of ?arg_1 the first occurrence of a sequence of collation units that provides a minimal match to the collation units of ?arg_2 according to the collation that is used.

  If any argument value is outside of its domain, the value of the function is left unspecified.

3.5.1.13 `func:replace` (adapted from `fn:replace`)

- *Schema*:

```
( ?input ?pattern ?replacement;
func:replace1( ?input ?pattern ?replacement ) )
```

```
( ?input ?pattern ?replacement ?flags;
func:replace2( ?input ?pattern ?replacement ?flags ) )
```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  The function returns the `xs:string` that is obtained by replacing each non-overlapping substring of ?input that matches the given ?pattern with an occurrence of the ?replacement string.

  If any argument value is outside of its domain, the value of the function is left unspecified.

3.5.2 Predicates on Strings

3.5.2.1 `pred:contains` (adapted from `fn:contains`)

- *Schema*:

  ```
  ( ?arg1 ?arg2; pred:contains1( ?arg1 ?arg2 ) )

  ( ?arg1 ?arg2 ?collation ;
  pred:contains2( ?arg1 ?arg2 ?collation ) )
  ```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping:*

  When all arguments belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:contains1(?arg1 ?arg2))(s_1 s_2) = **t** if and only if fn:contains(s_1, s_2) returns `true`, as defined in [XPath-Functions], analogously for pred:contains2. I.e., this function returns true or false indicating whether or not ?s_1 contains (at the beginning, at the end, or anywhere within) at least one sequence of collation units that provides a minimal match to the collation units in the value of ?s_2, according to the collation that is used. "Minimal match" is defined in Unicode Collation Algorithm.

  If an argument value is outside of its domain, the truth value of the function is left unspecified.

The following schemata are defined analogously with respect to their corresponding functions as defined in [XPath-Functions] and we only give informal descriptions of the respective mappings $I_{truth} \circ I_{external}$.

3.5.2.2 `pred:starts-with` (adapted from `fn:starts-with`)

- *Schema*:

  ```
  ( ?arg1 ?arg2; pred:starts-with1( ?arg1 ?arg2 )

  ( ?arg1 ?arg2 ?collation; pred:starts-
  with2( ?arg1 ?arg2 ?collation)
  ```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns true or false indicating whether or not the value of ?arg$_1$ starts with a sequence of collation units that provides a minimal match to the collation units of ?arg$_2$ according to the collation that is used.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.5.2.3 `pred:ends-with` (adapted from `fn:ends-with`)

- *Schema*:

  ```
  (?arg1 ?arg2; fn:ends-with1( ?arg1 ?arg2 ) )

  (?arg1 ?arg2 ?collation; fn:ends-
  with2( ?arg1 ?arg2 ?collation) )
  ```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns true or false indicating whether or not the value of ?arg$_1$ ends with a sequence of collation units that provides a minimal match to the collation units of ?arg$_2$ according to the collation that is used.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.5.2.4 `pred:matches` (adapted from `fn:matches`)

- *Schema*:

  ```
  ( ?input ?pattern; pred:matches1( ?input ?pattern) )

  ( ?input ?pattern ?flags;
  pred:matches2( ?input ?pattern ?flags ) )
  ```

- *Domains*:

  The value space of `xs:string` for all arguments.

- *Mapping (informal):*

  Returns true if the input matches the regular expression supplied as pattern as influenced by the flags, if present; otherwise, it returns false. The effect of calling the first version of this function (omitting the flags) is the same as the effect of calling the second version with the flags argument set to a zero-length string.

  If an argument value is outside of its domain, the value of the function is left unspecified.

Following the convention of having separate equality, inequality, less-than, greater-than, less-than-or-equal, greater-than-or-equal predicates for other datatypes, RIF defines such predicates as syntactic sugar over `func:compare` also for `xs:string` in the following.

Editor's Note: The need of separate less-than, greater-than, less-than-or-equal, greater-than-or-equal predicates for strings is still under discussion, cf. ISSUE-67.

3.5.2.5 `pred:string-equal`

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; pred:string-
  equal(?comparand1 ?comparand2) )
  ```

- *Domains*:

  The value space of `xs:string` for both arguments.

- *Mapping:*

When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:string-equal(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($ (?comparand$_1$?comparand$_2$; func:compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = 0.

If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

3.5.2.6 `pred:string-less-than`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:string-less-than(?comparand`$_1$ `?comparand`$_2$`) )`

- *Domains*:

  The value space of `xs:string` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:string-less-than(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($ (?comparand$_1$?comparand$_2$; func:compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = -1.

  If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

3.5.2.7 `pred:string-greater-than`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:string-greater-than(?comparand`$_1$ `?comparand`$_2$`) )`

- *Domains*:

  The value space of `xs:string` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:string-greater-than(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($

(?comparand$_1$?comparand$_2$; func:compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = 1.

If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

The following built-in predicates `pred:string-not-equal`, `pred:string-less-than-or-equal` and `pred:string-greater-than-or-equal` are defined accordingly.

3.5.2.8 `pred:string-not-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:string-not-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:string-not-equal` has the same domains as `pred:string-equal` and is true whenever `pred:string-equal` is false.

3.5.2.9 `pred:string-less-than-or-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:string-less-than-or-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:string-less-than-or-equal` has the same domains as `pred:string-equal` and is true whenever `pred:string-equal` is true or `pred:string-less-than` is true.

3.5.2.10 `pred:string-greater-than-or-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:string-greater-than-or-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:string-greater-than-or-equal` has the same domains as `pred:string-equal` and is true whenever `pred:string-equal` is true or `pred:string-greater-than` is true.

3.6 Functions and Predicates on Dates, Times, and Durations

If not stated otherwise, in the following we define schemas for functions and operators defined on the date, time and duration datatypes in [XPath-Functions].

As defined in Section 3.3.2 Dates and Times, `xs:dateTime`, `xs:date`, `xs:time`, `xs:gYearMonth`, `xs:gYear`, `xs:gMonthDay`, `xs:gMonth`, `xs:gDay` values, referred to collectively as date/time values, are represented as seven components or properties: year, month, day, hour, minute, second and timezone. The value of the first five components are `xs:integer`s. The value of the second component is an `xs:decimal` and the value of the timezone component is an `xs:dayTimeDuration`. For all the date/time datatypes, the timezone property is optional and may or may not be present. Depending on the datatype, some of the remaining six properties must be present and some must be absent. Absent, or missing, properties are represented by the empty sequence. This value is referred to as the local value in that the value is in the given timezone. Before comparing or subtracting `xs:dateTime` values, this local value must be translated or normalized to UTC.

3.6.1 Functions on Dates, Times, and Durations

3.6.1.1 `func:year-from-dateTime` (adapted from `fn:year-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:year-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping:*

  $I_{external}$(?arg ; func:year-from-dateTime(?arg))(s) = *res*

  such that *res* is the result of fn:year-from-dateTime(s) as defined in [XPath-Functions].

  If an argument value is outside of its domain, the value of the function is left unspecified.

Note that we we slightly deviate here from the original definition of fn:year-from-dateTime which says: "If ?arg is the empty sequence, returns the empty sequence." RIF predicates ad functions do not support "sequences". The following schemata

are defined analogously with respect to their corresponding operators as defined in [XPath-Functions] and we only give informal descriptions of the respective mappings $I_{external}$.

3.6.1.2 `func:month-from-dateTime` (adapted from `fn:month-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:month-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 1 and 12, both inclusive, representing the month component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.3 `func:day-from-dateTime` (adapted from `fn:day-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:day-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 1 and 31, both inclusive, representing the day component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.4 `func:hours-from-dateTime` (adapted from `fn:hours-from-dateTime`)

- *Schema*:

```
( ?arg ; func:hours-from-dateTime( ?arg ) )
```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 0 and 23, both inclusive, representing the hours component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.5 `func:minutes-from-dateTime` (adapted from `fn:minutes-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:minutes-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 0 and 59, both inclusive, representing the minutes component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.6 `func:seconds-from-dateTime` (adapted from `fn:seconds-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:seconds-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime` for ?arg.

- *Mapping (informal):*

Returns an `xs:decimal` value greater than or equal to zero and less than 60, representing the seconds and fractional seconds in the localized value of ?arg.

If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.7 `func:year-from-date` (adapted from `fn:year-from-date`)

- *Schema*:

  ```
  ( ?arg ; func:year-from-date( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:date` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` representing the year in the localized value of ?arg. The value may be negative.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.8 `func:month-from-date` (adapted from `fn:month-from-date`)

- *Schema*:

  ```
  ( ?arg ; func:month-from-date( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:date` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 1 and 12, both inclusive, representing the month component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.9 `func:day-from-date` (adapted from `fn:day-from-date`)

- *Schema*:

```
( ?arg ; func:day-from-date( ?arg ) )
```

- *Domain*:

  The value space of `xs:date` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 1 and 31, both inclusive, representing the day component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.10 `func:hours-from-time` (adapted from `fn:hours-from-time`)

- *Schema*:

  ```
  ( ?arg ; func:hours-from-time( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:time` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 0 and 23, both inclusive, representing the hours component in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.11 `func:minutes-from-time` (adapted from `fn:minutes-from-time`)

- *Schema*:

  ```
  ( ?arg ; func:minutes-from-time( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:time` for ?arg.

- *Mapping (informal):*

  Returns an `xs:integer` between 0 and 59, both inclusive, representing the minutes component in the localized value of ?arg.

If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.12 `func:seconds-from-time` (adapted from `fn:seconds-from-time`)

- *Schema*:

  ```
  ( ?arg ; func:seconds-from-time( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:time` for ?arg.

- *Mapping (informal):*

  Returns an `xs:decimal` value greater than or equal to zero and less than 60, representing the seconds and fractional seconds in the localized value of ?arg.

  If an argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.13 `func:years-from-duration` (adapted from `fn:years-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:years-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:yearMonthDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:integer` representing the years component in the value of ?arg. The result is obtained by casting ?arg to an `xs:yearMonthDuration` and then computing the years component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.14 `func:months-from-duration` (adapted from `fn:months-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:months-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:yearMonthDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:integer` representing the months component in the value of ?arg. The result is obtained by casting ?arg to an `xs:yearMonthDuration` and then computing the months component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.15 `func:days-from-duration` (adapted from `fn:days-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:days-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:dayTimeDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:integer` representing the days component in the value of ?arg. The result is obtained by casting ?arg to an `xs:dayTimeDuration` and then computing the days component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.16 `func:hours-from-duration` (adapted from `fn:hours-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:hours-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:dayTimeDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:integer` representing the hours component in the value of ?arg. The result is obtained by casting ?arg to an `xs:dayTimeDuration` and then computing the hours component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.17 `func:minutes-from-duration` (adapted from `fn:minutes-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:minutes-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:dayTimeDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:integer` representing the minutes component in the value of ?arg. The result is obtained by casting ?arg to an `xs:dayTimeDuration` and then computing the minutes component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.18 `func:seconds-from-duration` (adapted from `fn:seconds-from-duration`)

- *Schema*:

  ```
  ( ?arg ; func:seconds-from-duration( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of datatypes castable to `xs:dayTimeDuration` according to the table in Section 17.1 of [XPath-Functions].

- *Mapping (informal):*

  Returns an `xs:decimal` representing the seconds component in the value of ?arg. The result is obtained by casting ?arg to an `xs:dayTimeDuration` and then computing the seconds component.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.6.1.19 `func:timezone-from-dateTime` (adapted from `fn:timezone-from-dateTime`)

- *Schema*:

  ```
  ( ?arg ; func:timezone-from-dateTime( ?arg ) )
  ```

- *Domain*:

  The value space of `xs:dateTime`.

- *Mapping (informal):*

  Returns the timezone component of ?arg if any. If $arg has a timezone component, then the result is an `xs:dayTimeDuration` that indicates deviation from UTC; its value may range from +14:00 to -14:00 hours, both inclusive.

  If the argument value is outside of its domain, the value of the function is left unspecified.

The following two functions are defined analogously for domains `xs:date` and `xs:time`

3.6.1.20 `func:timezone-from-date` (adapted from `fn:timezone-from-date`)

- *Schema*:

```
( ?arg ; func:timezone-from-date( ?arg ) )
```

3.6.1.21 `func:timezone-from-time` (adapted from `fn:timezone-from-time`)

- *Schema*:

```
( ?arg ; func:timezone-from-time( ?arg ) )
```

3.6.1.22 `func:subtract-dateTimes` (adapted from `op:subtract-dateTimes`)

- *Schema*:

```
( ?arg1 ?arg2; func:subtract-dateTimes( ?arg1 ?arg2 ) )
```

Subtracts two `xs:dateTime`s. Returns an `xs:xs:dayTimeDuration`.

3.6.1.23 `func:subtract-dates` (adapted from `op:subtract-dates`)

- *Schema*:

```
( ?arg1 ?arg2; func:subtract-dates( ?arg1 ?arg2 ) )
```

Subtracts two `xs:date`s. Returns an `xs:xs:dayTimeDuration`.

3.6.1.24 `func:subtract-times` (adapted from `op:subtract-times`)

- *Schema*:

```
( ?arg1 ?arg2; func:subtract-times( ?arg1 ?arg2 ) )
```

Subtracts two `xs:time`s. Returns an `xs:xs:dayTimeDuration`.

3.6.1.25 `func:add-yearMonthDurations` (adapted from `op:add-yearMonthDurations`)

- *Schema*:

```
( ?arg1 ?arg2; func:add-yearMonthDurations( ?arg1 ?arg2
) )
```

Adds two `xs:yearMonthDuration`s. Returns an `xs:yearMonthDuration`.

3.6.1.26 `func:subtract-yearMonthDurations` (adapted from `op:subtract-yearMonthDurations`)

- *Schema*:

```
( ?arg1 ?arg2; func:subtract-
yearMonthDurations( ?arg1 ?arg2 ) )
```

Subtracts one `xs:yearMonthDuration` from another. Returns an `xs:yearMonthDuration`.

3.6.1.27 `func:multiply-yearMonthDuration` (adapted from `op:multiply-yearMonthDuration`)

- *Schema*:

```
( ?arg1 ?arg2; func:multiply-
yearMonthDuration( ?arg1 ?arg2 ) )
```

Multiplies an `xs:yearMonthDuration` by an `xs:double`. Returns an `xs:yearMonthDuration`.

3.6.1.28 `func:divide-yearMonthDuration` (adapted from `op:divide-yearMonthDuration`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:divide-
yearMonthDuration( ?arg1 ?arg2 ) )
```

Divides an `xs:yearMonthDuration` by an `xs:double`. Returns an `xs:yearMonthDuration`.

3.6.1.29 `func:divide-by-yearMonthDuration` (adapted from `op:divide-yearMonthDuration-by-yearMonthDuration`)

- *Schema*:

```
( ?arg1 ?arg2; func:divide-yearMonthDuration-by-
yearMonthDuration( ?arg1 ?arg2 ) )
```

Divides an `xs:yearMonthDuration` by an `xs:yearMonthDuration`. Returns an `xs:decimal`.

3.6.1.30 `func:add-dayTimeDurations` (adapted from `op:add-dayTimeDurations`)

- *Schema*:

```
( ?arg1 ?arg2; func:add-dayTimeDurations( ?arg1 ?arg2 )
)
```

Adds two `xs:dayTimeDuration`s. Returns an `xs:dayTimeDuration`.

3.6.1.31 `func:subtract-dayTimeDurations` (adapted from `op:subtract-dayTimeDurations`)

- *Schema*:

```
( ?arg1 ?arg2; func:subtract-
dayTimeDurations( ?arg1 ?arg2 ) )
```

Subtracts one `xs:dayTimeDuration` from another. Returns an `xs:dayTimeDuration`.

3.6.1.32 `func:multiply-dayTimeDuration` (adapted from `op:multiply-dayTimeDuration`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:multiply-
dayTimeDuration( ?arg1 ?arg2 ) )
```

Multiplies an `xs:dayTimeDuration` by a `xs:double`. Returns an `xs:dayTimeDuration`.

3.6.1.33 `func:divide-dayTimeDuration` (adapted from `op:divide-dayTimeDuration`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:divide-
dayTimeDuration( ?arg1 ?arg2 ) )
```

Divides an `xs:dayTimeDuration` by an `xs:double`. Returns an `xs:dayTimeDuration`.

3.6.1.34 `func:divide-dayTimeDuration-by-dayTimeDuration` (adapted from `op:divide-dayTimeDuration-by-dayTimeDuration`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:divide-dayTimeDuration-by-
dayTimeDuration( ?arg1 ?arg2 ) )
```

Divides an `xs:dayTimeDuration` by an `xs:dayTimeDuration`. Returns an `xs:decimal`.

3.6.1.35 `func:add-yearMonthDuration-to-dateTime` (adapted from `op:add-yearMonthDuration-to-dateTime`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:add-yearMonthDuration-to-
dateTime( ?arg1 ?arg2 ) )
```

Adds a `xs:yearMonthDuration` (`?arg2`) to a `xs:dateTime` (`?arg1`). Returns an `xs:dateTime`.

3.6.1.36 `func:add-yearMonthDuration-to-date` (adapted from `op:add-yearMonthDuration-to-date`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:add-yearMonthDuration-to-
date( ?arg1 ?arg2 ) )
```

Adds a `xs:yearMonthDuration` (`?arg2`) to a `xs:date` (`?arg1`). Returns an `xs:date`.

3.6.1.37 `func:add-dayTimeDuration-to-dateTime` (adapted from `op:add-dayTimeDuration-to-dateTime`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:add-dayTimeDuration-to-
dateTime( ?arg1 ?arg2 ) )
```

Adds a `xs:dayTimeDuration` (`?arg2`) to a `xs:dateTime` (`?arg1`). Returns an `xs:dateTime`.

3.6.1.38 `func:add-dayTimeDuration-to-date` (adapted from `op:add-dayTimeDuration-to-date`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:add-dayTimeDuration-to-
date( ?arg1 ?arg2 ) )
```

Adds a `xs:dayTimeDuration` (`?arg2`) to a `xs:date` (`?arg1`). Returns an `xs:date`.

3.6.1.39 `func:add-dayTimeDuration-to-time` (adapted from `op:add-dayTimeDuration-to-time`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:add-dayTimeDuration-to-
time( ?arg1 ?arg2 ) )
```

Adds a `xs:dayTimeDuration` (`?arg2`) to a `xs:time` (`?arg1`). Returns an `xs:time`.

3.6.1.40 `func:subtract-yearMonthDuration-from-dateTime` (adapted from `op:subtract-yearMonthDuration-from-dateTime`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:subtract-yearMonthDuration-from-
dateTime( ?arg1 ?arg2 ) )
```

Subtracts a `xs:yearMonthDuration` (`?arg2`) from a `xs:dateTime` (`?arg1`). Returns an `xs:dateTime`.

3.6.1.41 `func:subtract-yearMonthDuration-from-date` (adapted from `op:subtract-yearMonthDuration-from-date`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:subtract-yearMonthDuration-from-
date( ?arg1 ?arg2 ) )
```

Subtracts a `xs:yearMonthDuration` (`?arg2`) from a `xs:date` (`?arg1`). Returns an `xs:date`.

3.6.1.42 `func:subtract-dayTimeDuration-from-dateTime` (adapted from `op:subtract-dayTimeDuration-from-dateTime`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:subtract-dayTimeDuration-from-
dateTime( ?arg1 ?arg2 ) )
```

Subtracts a `xs:dayTimeDuration` (`?arg2`) from a `xs:dateTime` (`?arg1`). Returns an `xs:dateTime`.

3.6.1.43 `func:subtract-dayTimeDuration-from-date` (adapted from `op:subtract-dayTimeDuration-from-date`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:subtract-dayTimeDuration-from-
date( ?arg1 ?arg2 ) )
```

Subtracts a `xs:dayTimeDuration` (`?arg2`) from a `xs:date` (`?arg1`). Returns an `xs:date`.

3.6.1.44 `func:subtract-dayTimeDuration-from-time` (adapted from `op:subtract-dayTimeDuration-from-time`)

- *Schema*:

```
( ?arg1 ?arg2 ; func:subtract-dayTimeDuration-from-
time( ?arg1 ?arg2 ) )
```

Subtracts a `xs:dayTimeDuration` (`?arg2`) from a `xs:time` (`?arg1`). Returns an `xs:time`.

3.6.2 Predicates on Dates, Times, and Durations

3.6.2.1 `pred:dateTime-equal` (adapted from `op:dateTime-equal`)

- *Schema*:

```
( ?arg1 ?arg2; pred:dateTime-equal( ?arg1 ?arg2) )
```

- *Domains*:

  The value space of `xs:dateTime` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:dateTime-equal(?arg1 ?arg2))(s_1 s_2) = **t**

  if and only if op:dateTime-equal(s_1, s_2) returns `true`, as defined in [XPath-Functions], **f** in case `false` is returned.

  If an argument value is outside of its domain, the truth value of the function is left unspecified.

The following schemata for comparison operators are defined analogously with respect to their corresponding operators as defined in [XPath-Functions], where the domain for both arguments is implicit by the operator name and we only give additional details on domains and mapping as needed.

3.6.2.2 `pred:dateTime-less-than` (adapted from `op:dateTime-less-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:dateTime-less-than(?arg1 ?arg2 ) )
```

3.6.2.3 `pred:dateTime-greater-thanl` (adapted from `op:dateTime-greater-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:dateTime-greater-than(?arg1 ?arg2 )
)
```

3.6.2.4 `pred:date-equal` (adapted from `op:date-equal`)

- *Schema*:

```
( ?arg1 ?arg2; pred:date-equal( ?arg1 ?arg2) )
```

3.6.2.5 `pred:date-less-than` (adapted from `op:date-less-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:date-less-than(?arg1 ?arg2 ) )
```

3.6.2.6 `pred:date-greater-than` (adapted from `op:date-greater-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:date-greater-than(?arg1 ?arg2 ) )
```

3.6.2.7 `pred:time-equal` (adapted from `op:time-equal`)

- *Schema*:

```
( ?arg1 ?arg2; pred:time-equal( ?arg1 ?arg2) )
```

3.6.2.8 `pred:time-less-than` (adapted from `op:time-less-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:time-less-than(?arg1 ?arg2 ) )
```

3.6.2.9 `pred:time-greater-than` (adapted from `op:time-greater-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:time-greater-than(?arg1 ?arg2 ) )
```

3.6.2.10 `pred:duration-equal` (adapted from `op:duration-equal`)

- *Schema*:

```
( ?arg1 ?arg2; pred:duration-equal( ?arg1 ?arg2) )
```

- *Domains*:

  The union of the value spaces of `xs:dayTimeDuration` and `xs:yearMonthDuration` for both arguments.

3.6.2.11 `pred:dateTimeDuration-less-than` (adapted from `op:dayTimeDuration-less-than`)

- *Schema*:

```
( ?arg1 ?arg2; pred:dayTimeDuration-less-
than(?arg1 ?arg2 ) )
```

3.6.2.12 `pred:dayTimeDuration-greater-than` (adapted from `op:dayTimeDuration-greater-than`)

- *Schema*:

  (?arg$_1$?arg$_2$; pred:dayTimeDuration-greater-than(?arg$_1$?arg$_2$))

3.6.2.13 `pred:yearMonthDuration-less-than` (adapted from `op:yearMonthDuration-less-than`)

- *Schema*:

  (?arg$_1$?arg$_2$; pred:yearMonthDuration-less-than(?arg$_1$?arg$_2$))

3.6.2.14 `pred:yearMonthDuration-greater-than` (adapted from `op:yearMonthDuration-greater-than`)

- *Schema*:

  (?arg$_1$?arg$_2$; pred:yearMonthDuration-greater-than(?arg$_1$?arg$_2$))

3.6.2.15 `pred:dateTime-not-equal`

- *Schema*:

  (?arg$_1$?arg$_2$; pred:dateTime-not-equal(?arg$_1$?arg$_2$))

The predicate `pred:dateTime-not-equal` has the same domains as `pred:dateTime-equal` and is true whenever `pred:dateTime-equal` is false.

3.6.2.16 `pred:dateTime-less-than-or-equal`

- *Schema*:

  (?arg$_1$?arg$_2$; pred:dateTime-less-than-or-equal(?arg$_1$?arg$_2$))

The predicate `pred:dateTime-less-than-or-equal` has the same domains as `pred:dateTime-equal` and is true whenever `pred:dateTime-equal` is true or `pred:dateTime-less-than` is true.

3.6.2.17 `pred:dateTime-greater-than-or-equal`

- *Schema*:

  `(?arg1 ?arg2; pred:dateTime-greater-than-or-equal( ?arg1 ?arg2) )`

The predicate `pred:dateTime-greater-than-or-equal` has the same domains as `pred:dateTime-equal` and is true whenever `pred:dateTime-equal` is true or `pred:dateTime-greater-than` is true.

3.6.2.18 `pred:date-not-equal`

- *Schema*:

  `(?arg1 ?arg2; pred:date-not-equal( ?arg1 ?arg2) )`

The predicate `pred:date-not-equal` has the same domains as `pred:date-equal` and is true whenever `pred:date-equal` is false.

3.6.2.19 `pred:date-less-than-or-equal`

- *Schema*:

  `(?arg1 ?arg2; pred:date-less-than-or-equal( ?arg1 ?arg2) )`

The predicate `pred:date-less-than-or-equal` has the same domains as `pred:date-equal` and is true whenever `pred:date-equal` is true or `pred:date-less-than` is true.

3.6.2.20 `pred:date-greater-than-or-equal`

- *Schema*:

  `(?arg1 ?arg2; pred:date-greater-than-or-equal( ?arg1 ?arg2) )`

The predicate `pred:date-greater-than-or-equal` has the same domains as `pred:date-equal` and is true whenever `pred:date-equal` is true or `pred:date-greater-than` is true.

3.6.2.21 `pred:time-not-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:time-not-equal( ?arg1 ?arg2) )
```

The predicate `pred:time-not-equal` has the same domains as `pred:time-equal` and is true whenever `pred:time-equal` is false.

3.6.2.22 `pred:time-less-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:time-less-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:time-less-than-or-equal` has the same domains as `pred:time-equal` and is true whenever `pred:time-equal` is true or `pred:time-less-than` is true.

3.6.2.23 `pred:time-greater-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:time-greater-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:time-greater-than-or-equal` has the same domains as `pred:time-equal` and is true whenever `pred:time-equal` is true or `pred:time-greater-than` is true.

3.6.2.24 `pred:duration-not-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:duration-not-equal( ?arg1 ?arg2) )
```

The predicate `pred:duration-equal` has the same domains as `pred:duration-equal` and is true whenever `pred:duration-equal` is false.

Editor's Note: The introduction of less-than-or-eaual and greater-than-or-equal predicates for dayTimeDuration and yearMonthDuration still needs a WG resolution.

3.6.2.25 `pred:dayTimeDuration-less-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:dayTimeDuration-less-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:dayTimeDuration-less-than-or-equal` has the same domains as `pred:dayTimeDuration-less-than` and is true whenever `pred:duration-equal` is true or `pred:dayTimeDuration-less-than` is true.

3.6.2.26 `pred:dayTimeDuration-greater-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:dayTimeDuration-greater-
than( ?arg1 ?arg2) )
```

The predicate `pred:dayTimeDuration-greater-than-or-equal` has the same domains as `pred:dayTimeDuration-greater-than` and is true whenever `pred:duration-equal` is true or `pred:dayTimeDuration-greater-than` is true.

3.6.2.27 `pred:yearMonthDuration-less-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:yearMonthDuration-less-than-or-
equal( ?arg1 ?arg2) )
```

The predicate `pred:yearMonthDuration-less-than-or-equal` has the same domains as `pred:yearMonthDuration-less-than` and is true whenever `pred:duration-equal` is true or `pred:yearMonthDuration-less-than` is true.

3.6.2.28 `pred:yearMonthDuration-greater-than-or-equal`

- *Schema*:

```
(?arg1 ?arg2; pred:yearMonthDuration--greater-
than( ?arg1 ?arg2) )
```

The predicate `pred:yearMonthDuration-greater-than-or-equal` has the same domains as `pred:yearMonthDuration-greater-than` and is true whenever `pred:duration-equal` is true or `pred:yearMonthDuration-greater-than` is true.

3.7 Functions and Predicates on rdf:XMLLiterals

Editor's Note: Predicates for `rdf:XMLLiteral` such as at least comparison predicates (equals, not-equals) are still under discussion in the working group.

3.7.1 `pred:XMLLiteral-equal`

- *Schema*:

  (?arg$_1$?arg$_2$; pred:XMLLiteral-equal(?arg$_1$?arg$_2$))

- *Domains*:

  The value space of `rdf:XMLLiteral` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg$_1$?arg$_2$; pred:XMLLiteral-equal(?arg$_1$?arg$_2$))(s_1 s_2) = **t**

  if and only if $s_1 = s_2$, **f** otherwise.

  If an argument value is outside of its domain, the truth value of the function is left unspecified.

3.7.2 `pred:XMLLiteral-not-equal`

- *Schema*:

  (?arg$_1$?arg$_2$; pred:XMLLiteral-not-equal(?arg$_1$?arg$_2$))

The predicate `pred:time-not-equal` has the same domains as `pred:XMLLiteral-equal` and is true whenever `pred:XMLLiteral-equal` is false.

3.8 Functions and Predicates on rdf:text

The following functions and predicates are adapted from the respective functions and operators in [RDF-TEXT].

Editor's Note: Issues which are still open in the rdf:text specification might imply future changes on the functions and predicates defined here. For instance reuse

of the fn: namespace is still under discussion, cf. http://lists.w3.org/Archives/Public/public-rdf-text/2008OctDec/0020.html. Moreover, references and links to the [RDF-TEXT] document will be updated in future versions of this document.

3.8.1 Functions on rdf:text

3.8.1.1 `func:text-from-string-lang` (adapted from `fn:text-from-string-lang`)

- *Schema*:

  `(?arg1 ?arg2 ; func:text-from-string-lang( ?arg1 ?arg2 ) )`

- *Domains*:

  The value space of `xs:string` for ?arg1 and the intersection of the elements of the value space of `xs:string` which represent valid language tags according to [BCP-47] for ?arg2.

- *Mapping:*

  $I_{external}$((?arg1 ?arg2 ; func:text-from-string-lang(?arg1 ?arg2))(s l) = *res* such that *res* is the pair (s, l) in the value space of `rdf:text`.

  If any argument value is outside of its domain, the value of the function is left unspecified.

3.8.1.2 `func:text-from-string` (adapted from `fn:text-from-string`)

- *Schema*:

  `(?arg ; func:text-from-string( ?arg ) )`

- *Domain*:

  The value space of `xs:string` for ?arg.

- *Mapping:*

  $I_{external}$((?arg ; func:text-from-string(?arg))(s) = *res* such that *res* is the pair (s, "") in the value space of `rdf:text`.

  If the argument value is outside of its domain, the value of the function is left unspecified.

Note: Since RIF implementations MAY choose to interpret `xs:string` and its subtypes as subtypes of `rdf:text` following Section 3.1 of [RDF-TEXT], in such implementations this function may just be implemented as the identity function.

3.8.1.3 `func:string-from-text` (adapted from `fn:string-from-text`)

- *Schema*:

  ```
  (?arg ; func:string-from-text( ?arg ) )
  ```

- *Domain*:

  The value space of `rdf:text` for ?arg.

- *Mapping:*

  $I_{external}$(?arg ; func:string-from-text(?arg))(t) = *res* such that *res* is the string part s of t if t=(s, l) is in the domain of ?arg.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.8.1.4 `func:lang-from-text` (adapted from `fn:lang-from-text`)

- *Schema*:

  ```
  (?arg ; func:lang-from-text( ?arg ) )
  ```

- *Domain*:

  The value space of `rdf:text` for ?arg.

- *Mapping:*

  $I_{external}$(?arg ; func:lang-from-text(?arg))(t) = *l* such that *l* is the language tag string of t if t= (s, l) is in the domain of ?arg and `""^^xs:string` otherwise.

3.8.1.5 `func:text-compare` (adapted from `fn:text-compare`)

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; func:text-
  compare1(?comparand1 ?comparand2) )
  ```

`( ?comparand1 ?comparand2 ?collation; func:text-compare2(?comparand1 ?comparand2 ?collation) )`

- *Domains*:

  The value space of `rdf:text` for `?comparand1` and `$comparand2`, and the value space of `xs:string` for `?collation`.

- *Mapping:*

  $I_{external}$((?comparand1 ?comparand2; func:text-compare1(?comparand1 ?comparand2))(t_1 t_2) = *res* such that, whenever t_1=(s_1, l) and t_2=(s_2, l) are two pairs with the same language tag l in the value space of`rdf:text`, *res* = `-1`, `0`, or `1` (from the value space of `xs:integer`), depending on whether the value of s_1 is respectively less than, equal to, or greater than the value of s_2 according to the rules of the collation that is used.

  In case an argument value is outside of its domain, or if the language tags of the values for `?comparand1` and `$comparand2` differ, the function value is left unspecified.

3.8.1.6 `func:text-length` (adapted from `fn:text-length`)

- *Schema*:

  `( func:text-length1() )`

  `( ?arg ; func:text-length2( ?arg ) )`

- *Domain*:

  The value space of `rdf:text` for ?arg.

- *Mapping:*

  Returns an `xs:integer` equal to the length in characters of the string part *s* of the argument if it is a pair (*s*,l) in the value space of `rdf:text`, returns 0 when called without an argument.

  If the argument value is outside of its domain, the value of the function is left unspecified.

3.8.2 Predicates on rdf:text

3.8.2.1 `pred:matches-language-range` (adapted from `fn:matches-language-range`)

- *Schema*:

  ```
  ( ?input ?range; pred:matches-language-
  range( ?input ?range) )
  ```

- *Domains*:

  The value space of `rdf:text` for `?input` and the value space of `xs:string` for all `?range`.

- *Mapping:*

  Whenever both arguments are within their domains, returns true if and only if the language tag of `?input` is a valid language tag according to BCP-47 [BCP-47], and if it matches the language-range expression supplied as `?range` as specified by the algorithm for "Matching of Language Tags" which is part of BCP-47 [BCP-47]; otherwise, it returns false.

  If an argument value is outside of its domain, the truth value of the predicate is left unspecified.

Following the convention of having separate equality, inequality, less-than, greater-than, less-than-or-equal, greater-than-or-equal predicates for other datatypes, RIF defines such predicates as syntactic sugar over `func:text-compare` also for `rdf:text` in the following.

3.8.2.2 `pred:text-equal`

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; pred:text-
  equal(?comparand1 ?comparand2) )
  ```

- *Domains*:

  The value space of `rdf:text` for both arguments.

- *Mapping:*

When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:text-equal(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($ (?comparand$_1$?comparand$_2$; func:text-compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = 0.

If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

3.8.2.3 `pred:text-less-than`

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; pred:text-less-
  than(?comparand1 ?comparand2) )
  ```

- *Domains*:

  The value space of `rdf:text` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:text-less-than(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($ (?comparand$_1$?comparand$_2$; func:text-compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = -1.

  If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

3.8.2.4 `pred:text-greater-than`

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; pred:text-greater-
  than(?comparand1 ?comparand2) )
  ```

- *Domains*:

  The value space of `rdf:text` for both arguments.

- *Mapping:*

  When both s_1 and s_2 belong to their domains, $I_{truth} \circ I_{external}($ (?comparand$_1$?comparand$_2$; pred:text-greater-than(?comparand$_1$?comparand$_2$))(s_1 s_2) = **t** if and only if $I_{external}($

(?comparand$_1$?comparand$_2$; func:text-compare(?comparand$_1$?comparand$_2$))(s_1 s_2) = 1.

If an argument value is outside of its domain, the value of the truth value of the predicate is left unspecified.

The following built-in predicates `pred:text-not-equal`, `pred:text-less-than-or-equal` and `text:text-greater-than-or-equal` are defined accordingly.

3.8.2.5 `pred:text-not-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:text-not-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:text-not-equal` has the same domains as `pred:text-equal` and is true whenever `pred:text-equal` is false.

3.8.2.6 `pred:text-less-than-or-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:text-less-than-or-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:text-less-than-or-equal` has the same domains as `pred:text-equal` and is true whenever `pred:text-equal` is true or `pred:text-less-than` is true.

3.8.2.7 `pred:text-greater-than-or-equal`

- *Schema*:

  `( ?comparand`$_1$ `?comparand`$_2$`; pred:text-greater-than-or-equal(?comparand`$_1$ `?comparand`$_2$`) )`

The predicate `pred:text-greater-than-or-equal` has the same domains as `pred:text-equal` and is true whenever `pred:text-equal` is true or `pred:text-greater-than` is true.

Editor's Note: The need of separate less-than, greater-than, less-than-or-equal, greater-than-or-equal predicates for `rdf:text` is still under discussion, cf. ISSUE-67

4 References

[BCP-47]
BCP 47 - Tags for the Identification of Languages, A. Phillips, M. Davis, IETF, Sep 2006, ftp://ftp.rfc-editor.org/in-notes/bcp/bcp47.txt.

[BLD]
RIF Basic Logic Dialect Harold Boley, Michael Kifer, eds. W3C Working Draft, 30 July 2008, http://www.w3.org/TR/2008/WD-rif-bld-20080730/. Latest version available at http://www.w3.org/TR/rif-bld/.

[CURIE]
CURIE Syntax 1.0, M. Birbeck, S. McCarron, W3C Working Draft, 6 May 2008, http://www.w3.org/TR/2007/WD-curie-20071126/. Latest version available at http://www.w3.org/TR/curie/.

[RDF-CONCEPTS]
Resource Description Framework (RDF): Concepts and Abstract Syntax, G. Klyne, J. Carroll (Editors), W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Latest version available at http://www.w3.org/TR/rdf-concepts/.

[RDF-SEMANTICS]
RDF Semantics, P. Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/. Latest version available at http://www.w3.org/TR/rdf-mt/.

[RDF-SCHEMA]
RDF Vocabulary Description Language 1.0: RDF Schema, B. McBride, Editor, W3C Recommendation 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/. Latest version available at http://www.w3.org/TR/rdf-schema/.

[RDF-TEXT]
`rdf:text` *A Datatype for Internationalized Text* , J. Bao, A. Polleres, B. Motik (Editors), W3C Working Draft. Latest version available at http://www.w3.org/2007/OWL/wiki/InternationalizedStringSpec.

(**Reference will be adapted at publication time.**)

[RFC-3986]
RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, IETF, January 2005, http://www.ietf.org/rfc/rfc3986.txt.

[RFC-3987]
RFC 3987 - Internationalized Resource Identifiers (IRIs), M. Duerst and M. Suignard, IETF, January 2005, http://www.ietf.org/rfc/rfc3987.txt.

[SPARQL]
SPARQL Query Language for RDF, E. Prud'hommeaux, A. Seaborne (Editors), W3C Recommendation, World Wide Web Consortium, 12 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/. Latest version available at http://www.w3.org/TR/rdf-sparql-query/.

[XDM]
XQuery 1.0 and XPath 2.0 Data Model (XDM), M. Fernández, A. Malhotra, J. Marsh, M. Nagy, N. Walsh (Editors), W3C Recommendation, World Wide Web Consortium, 23 January 2007. This version is http://www.w3.org/TR/2007/REC-xpath-datamodel-20070123/. Latest version available at http://www.w3.org/TR/xpath-datamodel/.

[XML-NS]
Namespaces in XML 1.1 (Second Edition), T. Bray, D. Hollander, A. Layman, R. Tobin (Editors), W3C Recommendation, World Wide Web Consortium, 16 August 2006, http://www.w3.org/TR/2006/REC-xml-names11-20060816/. Latest version available at http://www.w3.org/TR/xml-names11/.

[XML-SCHEMA2]
XML Schema Part 2: Datatypes Second Edition, P. V. Biron, A. Malhotra (Editors), W3C Recommendation, World Wide Web Consortium, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/. Latest version available at http://www.w3.org/TR/xmlschema-2/.

[XPath-Functions]
XQuery 1.0 and XPath 2.0 Functions and Operators, A. Malhotra, J. Melton, N. Walsh (Editors), W3C Recommendation, World Wide Web Consortium, 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-functions-20070123/. Latest version available at http://www.w3.org/TR/xpath-functions/.

5 Appendix: Schemas for Externally Defined Terms

This section is an edited copy of a section from RIF Framework for Logic Dialects. It is reproduced here for convenience of readers familiar with the RIF-BLD document who might not have studied RIF-FLD.

This section defines *external schemas*, which serve as templates for externally defined terms. These schemas determine which externally defined terms are acceptable in a RIF dialect. Externally defined terms include RIF built-ins, but are more general. They are designed to also accommodate the ideas of procedural attachments and querying of external data sources. Because of the need to accommodate many difference possibilities, the RIF logical framework supports a very general notion of an externally defined term. Such a term is not necessarily a function or a predicate -- it can be a frame, a classification term, and so on.

Definition (Schema for external term). An ***external schema*** is a statement of the form `(?X1 ... ?Xn; τ)` where

- `τ` is a constant, a positional, named-argument, or a frame term.
- `?X1 ... ?Xn` is a list of all distinct variables that occur in `τ`

The names of the variables in an external schema are immaterial, but their order is important. For instance, `(?X ?Y; ?X[foo->?Y])` and `(?V ?W; ?V[foo->?W])` are considered to be indistinguishable, but `(?X ?Y; ?X[foo->?Y])` and `(?Y ?X; ?X[foo->?Y])` are viewed as different schemas.

A term `t` is an ***instance*** of an external schema `(?X1 ... ?Xn; τ)` iff `t` can be obtained from `τ` by a simultaneous substitution `?X1/s1 ... ?Xn/sn` of the variables `?X1 ... ?Xn` with terms `s1 ... sn`, respectively. Some of the terms `si` can be variables themselves. For example, `?Z[foo->f(a ?P)]` is an instance of `(?X ?Y; ?X[foo->?Y])` by the substitution `?X/?Z ?Y/f(a ?P).` □

Observe that a variable cannot be an instance of an external schema, since `τ` in the above definition cannot be a variable. It will be seen later that this implies that a term of the form `External(?X)` is not well-formed in RIF.

The intuition behind the notion of an external schema, such as `(?X ?Y; ?X["foo"^^xs:string->?Y])` or `(?V; "pred:isTime"^^rif:iri(?V))`, is that `?X["foo"^^xs:string->?Y]` or `"pred:isTime"^^rif:iri(?V)` are invocation patterns for querying external sources, and instances of those schemas correspond to concrete invocations. Thus, `External("http://foo.bar.com"^^rif:iri["foo"^^xs:string->"123"^^xs:integer])` and `External("pred:isTime"^^rif:iri("22:33:44"^^xs:time)` are examples of invocations of external terms -- one querying an external source and another invoking a built-in.

Definition (Coherent set of external schemas). A set of external schemas is ***coherent*** if there is no term, `t`, that is an instance of two distinct schemas in the set. □

The intuition behind this notion is to ensure that any use of an external term is associated with at most one external schema. This assumption is relied upon in the definition of the semantics of externally defined terms. Note that the coherence condition is easy to verify syntactically and that it implies that schemas like `(?X ?Y; ?X[foo->?Y])` and `(?Y ?X; ?X[foo->?Y])`, which differ only in the order of their variables, cannot be in the same coherent set.

It is important to keep in mind that external schemas are *not* part of the language in RIF, since they do not appear anywhere in RIF statements. Instead, they are best thought of as part of the grammar of the language.