W3C Candidate Recommendation

RIF Datatypes and Built-Ins 1.0

W3C Candidate Recommendation 1 October 2009

This version:
http://www.w3.org/TR/2009/CR-rif-dtb-20091001/
Latest version:
http://www.w3.org/TR/rif-dtb/
Previous version:
http://www.w3.org/TR/2009/WD-rif-dtb-20090703/ (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

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

W3C Candidate Recommendation

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May Be Superseded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 10 documents:

1. RIF Overview
2. RIF Core Dialect
3. RIF Basic Logic Dialect
4. RIF Framework for Logic Dialects
5. RIF RDF and OWL Compatibility
6. RIF Datatypes and Built-Ins 1.0 (this document)
7. RIF Production Rule Dialect
8. RIF Test Cases
9. RIF Combination with XML data
10. OWL 2 RL in RIF

Summary of Changes

There have been no substantive changes since the previous version. For details on the minor changes see the change log and color-coded diff.

Please Comment By 29 October 2009

The Rule Interchange Format (RIF) Working Group seeks to gather experience from implementations in order to increase confidence in the language and meet specific exit criteria. This document will remain a Candidate Recommendation until at least 29 October 2009. After that date, when and if the exit criteria are met, the group intends to request Proposed Recommendation status.

Please send reports of implementation experience, and other feedback, to public-rif-comments@w3.org (public archive). Reports of any success or difficulty with the test cases are encouraged. Open discussion among developers is welcome at public-rif-dev@w3.org (public archive).

W3C Candidate Recommendation

No Endorsement

Publication as a Candidate Recommendation 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.

Table of Contents

W3C Candidate Recommendation

1 Overview

This specification develops ***RIF-DTB*** (**D**atat**y**pes and **B**uilt-Ins of the **R**ule **I**nterchange **F**ormat). It lists the datatypes, built-in functions and built-in predicates expected to be supported by RIF dialects such as the RIF Core Dialect, the RIF Basic Logic Dialect, and the RIF Production Rules Dialect.

Some of the datatypes are adapted from [XML-SCHEMA2]. A large part of the definitions of the listed functions and operators are adapted from [XPath-Functions]. The `rdf:PlainLiteral` datatype as well as functions and operators associated with that datatype are adopted from [RDF-PLAINLITERAL]. Unlike the earlier SWRL built-ins [SWRL], which write n-ary functions as (1+n)-ary relations, functional RIF-DTB built-ins remain functions.

Currently in 1.0, RIF-DTB can also help in the interoperation of RIF with other (Semantic) Web formalisms by providing a general infrastructure of datatypes and built-ins.

2 Constants, Symbol Spaces, and Datatypes

2.1 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 the (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*.

2.2 The Base and Prefix Directives

Since IRIs typically require long strings of characters, many Web languages have special provisions for abbreviating these strings. One well-known 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 ***absolute*** 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***. The prefix p is an alphanumeric string and the expansion v 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 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`.

2.2.1 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.
- 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 symbol spaces listed in the following. However, rule sets that are exchanged through RIF can use additional symbol spaces.

In the following list we introduce ***short name***s for some of the symbol spaces. Short names are NCNames, typically the character sequence after the last '/' or '#' in the symbol space IRI (similar to the XML local name part of a QName). Short names are used for the predicates in Sections Guard Predicates for Datatypes and Negative Guard Predicates for Datatypes below.

- `xs:anyURI` (`http://www.w3.org/2001/XMLSchema#anyURI`), short name: `anyURI`
- `xs:base64Binary` (`http://www.w3.org/2001/XMLSchema#base64Binary`), short name: `base64Binary`
- `xs:boolean` (`http://www.w3.org/2001/XMLSchema#boolean`), short name: `boolean`
- `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:dateTimeStamp` (`http://www.w3.org/2001/XMLSchema#dateTimeStamp`), short name: `dateTimeStamp`
- `xs:double` (`http://www.w3.org/2001/XMLSchema#double`), short name: `double`
- `xs:float` (`http://www.w3.org/2001/XMLSchema#float`), short name: `float`
- `xs:hexBinary` (`http://www.w3.org/2001/XMLSchema#hexBinary`), short name: `hexBinary`
- `xs:decimal` (`http://www.w3.org/2001/XMLSchema#decimal`), short name: `decimal`
- `xs:integer` (`http://www.w3.org/2001/XMLSchema#integer`), short name: `integer`
- `xs:long` (`http://www.w3.org/2001/XMLSchema#long`), short name: `long`
- `xs:int` (`http://www.w3.org/2001/XMLSchema#int`), short name: `int`

- `xs:short` (`http://www.w3.org/2001/XMLSchema#short`), short name: `short`
- `xs:byte` (`http://www.w3.org/2001/XMLSchema#byte`), short name:`byte`
- `xs:nonNegativeInteger` (`http://www.w3.org/2001/XMLSchema#xs:nonNegativeInteger`), short name: `nonNegativeInteger`
- `xs:positiveInteger` (`http://www.w3.org/2001/XMLSchema#xs:positiveInteger`), short name: `positiveInteger`
- `xs:unsignedLong` (`http://www.w3.org/2001/XMLSchema#unsignedLong`), short name: `unsignedLong`
- `xs:unsignedInt` (`http://www.w3.org/2001/XMLSchema#unsignedInt`), short name: `unsignedInt`
- `xs:unsignedShort` (`http://www.w3.org/2001/XMLSchema#unsignedShort`), short name: `unsignedShort`
- `xs:unsignedByte` (`http://www.w3.org/2001/XMLSchema#unsignedByte`), short name: `unsignedByte`
- `xs:nonPositiveInteger` (`http://www.w3.org/2001/XMLSchema#xs:nonPositiveInteger`), short name: `nonPositiveInteger`
- `xs:negativeInteger` (`http://www.w3.org/2001/XMLSchema#negativeInteger`), short name: `negativeInteger`
- `xs:string` (`http://www.w3.org/2001/XMLSchema#string`), short name: `string`
- `xs:normalizedString` (`http://www.w3.org/2001/XMLSchema#xs:normalizedString`), short name: `normalizedString`
- `xs:token` (`http://www.w3.org/2001/XMLSchema#xs:token`), short name: `token`
- `xs:language` (`http://www.w3.org/2001/XMLSchema#language`), short name: `language`
- `xs:Name` (`http://www.w3.org/2001/XMLSchema#Name`), short name: `Name`
- `xs:NCName` (`http://www.w3.org/2001/XMLSchema#NCName`), short name: `NCName`
- `xs:NMTOKEN` (`http://www.w3.org/2001/XMLSchema#NMTOKEN`), short name: `NMTOKEN`
- `xs:time` (`http://www.w3.org/2001/XMLSchema#time`), short name: `time`

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`. The lexical spaces of the above symbol spaces are defined in the document [XDM].

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

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

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

The `rdf:XMLLiteral` 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`), for ***internationalized resource identifiers*** or ***IRI***s.

Constants in the `rif:iri` 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 `xs:anyURI`.

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

Constants in the `rif:local` 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***, where `X` is the (short) name of the respective symbol space. 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***.

2.2.2 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
                | '_' NCName                   // shortcut for "..."^^rif
                | '"' UNICODESTRING '"' '@' langtag             // shortc
```

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

- `IRI_REF`, cf. http://www.w3.org/TR/rdf-sparql-query/#rIRI_REF
- `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
- `NumericLiteral`, cf. http://www.w3.org/TR/rdf-sparql-query/#rNumericLiteral
- `NCName`, cf. http://www.w3.org/TR/2006/REC-xml-names11-20060816/#NT-NCName
- `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.
- `langtag`, cf. [BCP-47]

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.

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.

2.2.3 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>)
```

2.3 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 (Datatype). A ***datatype*** 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 datatypes. However, RIF dialects may include additional datatypes. Subitems in the following lists indicate derived datatypes.

- `xs:anyURI`
- `xs:base64Binary`
- `xs:boolean`
- `xs:date`
- `xs:dateTime`
    - `xs:dateTimeStamp`
- `xs:double`
- `xs:float`
- `xs:hexBinary`
- `xs:decimal`
    - `xs:integer`
        - `xs:long`
            - `xs:int`
                - `xs:short`
                    - `xs:byte`

        - xs:nonNegativeInteger
            - xs:positiveInteger
            - xs:unsignedLong
                - xs:unsignedInt
                    - xs:unsignedShort
                        - xs:unsignedByte
        - xs:nonPositiveInteger
            - xs:negativeInteger
- xs:string
    - xs:normalizedString
        - xs:token
            - xs:language
            - xs:Name
                - xs:NCName
            - xs:NMTOKEN
- xs:time
- xs:dayTimeDuration
- xs:yearMonthDuration
- rdf:PlainLiteral
- rdf:XMLLiteral

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

- For the XML Schema datatypes of RIF, namely all RIF datatypes within the `xs:` namespace, except `xs:dayTimeDuration` and `xs:yearMonthDuration`, 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 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:PlainLiteral` are defined in the document [RDF-PLAINLITERAL].
- The value space and lexical-to-value-space mapping for the datatype `rdf:XMLLiteral` is defined in RDF [RDF-CONCEPTS].

3 Syntax and Semantics of Built-ins

3.1 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 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

($?X_1$ ... $?X_n$; f($?X_1$ ... $?X_n$))

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

Note that RIF-BLD allows additional forms of built-ins, which includes named-argument terms.

RIF-FLD defines a very general notion of external terms and schemas, but RIF-BLD and the present document use more restricted notions. For convenience, we present a complete definition of these restricted notions in Appendix: Schemas for Externally Defined Terms.

3.2 Semantics of Built-ins

The semantics of external terms 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 $I_{truth} \circ I_{external}(\sigma)$.

4 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. The differences from the original [XPath-Functions] include the handling of errors, the differentiation between predicates and functions, and a few specific differences noted in the definitions below.

4.1 Predicates for all Datatypes

4.1.1 Comparison for Literals

RIF supports identity for typed literals through the "=" predicate in all dialects that extend RIF-Core. Identity for typed literals is defined as being the same point in the value space for that type. Certain datatypes use more specific notions of equality that allow for multiple points in the value space to be considered equal. For each datatype specific notion of equality we refer to the supported predicate for that datatype.

Since the basic RIF dialects do not support negation, dialects that extend RIF-Core define a built-in for checking the non-identity of two typed literals.

4.1.1.1 `pred:literal-not-identical`

- *Schema*:

  ```
  ( ?arg1 ?arg2; pred:literal-not-identical(  ?arg1 ?arg2 ) )
  ```

- *Domain*:

  This predicate does not depend on a specific domain.

- *Mapping:*

  $I_{truth} \circ I_{external}$(?arg$_1$?arg$_2$; pred:literal-not-identical(?arg$_1$?arg$_2$))(s_1 s_2) = **t** if and only if s_1 and s_2 are both in the value spaces of some datatypes in ***DTS*** and $s_1 \neq s_2$. This includes the case where s_1 and s_2 are of disjoint types.

  $I_{truth} \circ I_{external}$(?arg$_1$?arg$_2$; pred:literal-not-identical(?arg$_1$?arg$_2$))(s_1 s_2) = **f** otherwise. This includes the case where s_1 or s_2 are not in the value spaces of datatypes in ***DTS***.

4.2 Guard Predicates for Datatypes

RIF defines guard predicates for all datatypes in Section Datatypes.

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

  `( ?arg1 pred:is-literal-`*DATATYPE*` ( ?arg1 ) )`

  Here, *DATATYPE* is the short name for a datatype. For instance, we use `pred:is-literal-string` for the guard predicate for `xs:string`, `pred:is-literal-PlainLiteral` for the guard predicate for `rdf:PlainLiteral`, or `pred:is-literal-XMLLiteral` 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.

- *Domain*:

  Guard predicates do not depend on a specific domain.

- *Mapping:*

  $I_{truth} \circ I_{external}$(?arg1; pred:is-literal-*DATATYPE* (?arg1))(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.

- `( ?arg1; pred:is-literal-anyURI( ?arg1 ) )`
- `( ?arg1; pred:is-literal-base64Binary( ?arg1 ) )`
- `( ?arg1; pred:is-literal-boolean( ?arg1 ) )`
- `( ?arg1; pred:is-literal-date ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-dateTime ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-dateTimeStamp ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-double ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-float ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-hexBinary ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-decimal ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-integer( ?arg1 ) )`
- `( ?arg1; pred:is-literal-long ?arg1 ) )`
- `( ?arg1; pred:is-literal-int( ?arg1 ) )`
- `( ?arg1; pred:is-literal-short( ?arg1 ) )`
- `( ?arg1; pred:is-literal-byte( ?arg1 ) )`

- (?arg$_1$; pred:is-literal-nonNegativeInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-positiveInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-unsignedLong(?arg$_1$))
- (?arg$_1$; pred:is-literal-unsignedInt(?arg$_1$))
- (?arg$_1$; pred:is-literal-unsignedShort(?arg$_1$))
- (?arg$_1$; pred:is-literal-unsignedByte(?arg$_1$))
- (?arg$_1$; pred:is-literal-nonPositiveInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-negativeInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-PlainLiteral (?arg$_1$))
- (?arg$_1$; pred:is-literal-string (?arg$_1$))
- (?arg$_1$; pred:is-literal-normalizedString (?arg$_1$))
- (?arg$_1$; pred:is-literal-token (?arg$_1$))
- (?arg$_1$; pred:is-literal-language (?arg$_1$))
- (?arg$_1$; pred:is-literal-Name (?arg$_1$))
- (?arg$_1$; pred:is-literal-NCName (?arg$_1$))
- (?arg$_1$; pred:is-literal-NMTOKEN (?arg$_1$))
- (?arg$_1$; pred:is-literal-time (?arg$_1$))
- (?arg$_1$; pred:is-literal-dayTimeDuration (?arg$_1$))
- (?arg$_1$; pred:is-literal-yearMonthDuration (?arg$_1$))
- (?arg$_1$; pred:is-literal-XMLLiteral (?arg$_1$))

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

4.3 Negative Guard Predicates for Datatypes

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

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

  (?arg$_1$; pred:is-literal-not-*DATATYPE* (?arg$_1$))

  Here, *DATATYPE* is the short name for one of the datatypes mentioned in this document. For instance, we use `pred:is-literal-not-String` for the negative guard predicate for `xs:string`, `pred:is-literal-not-PlainLiteral` for the negative guard predicate for `rdf:PlainLiteral`, or `pred:is-literal-not-XMLLiteral` 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.

- *Domain*:

  Negative guard predicates do not depend on a specific domain.

- *Mapping:*

  $I_{truth} \circ I_{external}$(?arg$_1$; pred:is-literal-not-*DATATYPE* (?arg$_1$))(s$_1$) = **t** if and only if s$_1$ is in the value space of one of the datatypes in ***DTS*** but not in the value space of the datatype with shortname *DATATYPE*, and **f** otherwise.

Accordingly, the following schemas are defined.

- (?arg$_1$; pred:is-literal-not-anyURI(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-base64Binary(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-boolean(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-date (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-dateTime (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-dateTimeStamp (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-double (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-float (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-hexBinary (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-decimal (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-integer(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-long ?arg$_1$))
- (?arg$_1$; pred:is-literal-not-int(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-short(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-byte(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-nonNegativeInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-positiveInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-unsignedLong(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-unsignedInt(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-unsignedShort(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-unsignedByte(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-nonPositiveInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-negativeInteger(?arg$_1$))
- (?arg$_1$; pred:is-literal-not-PlainLiteral (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-string (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-normalizedString (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-token (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-language (?arg$_1$))
- (?arg$_1$; pred:is-literal-not-Name (?arg$_1$))

- `( ?arg1; pred:is-literal-not-NCName ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-not-NMTOKEN ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-not-time ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-not-dayTimeDuration ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-not-yearMonthDuration ( ?arg1 ) )`
- `( ?arg1; pred:is-literal-not-XMLLiteral ( ?arg1 ) )`

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

Note: The semantics of negative guards may be surprising. The is-literal-not-String guard essentially asks, "Is this a literal, *and* (if it is) is it something other than a String?" It could also be read as "Is this a decimal or a float or a double or a date or a dateTime, etc, [for every datatype except string] ?". The negative guards are formulated like this to allow for rules which detect, for instance, some kinds of bad inputs, while still using the open world assumption of some RIF dialects.

4.4 Datatype Conversion and Casting

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.

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.

4.4.1 Casting to XML Schema Datatypes

The casting functions in Section 17.1 of [XPath-Functions] define mappings from source values *SV*, which are data values, annotated with source types *ST*, to target values *TV*, annotated with target types *TT*. The data values *V* we consider are not necessarily explicitly annotated with types. However, one can view the datatypes $D_1,...,D_n$ whose value spaces include a data value *V* as the types of *V*. We assume in the following that any of the data types $D_1,...,D_n$ is used as the annotation of the source value *SV*; the conversions in [XPath-Functions] are defined equivalently for all such datatypes.

- *Schema*: The schemas for casting functions have the general form

  `( ?arg; DATATYPE-IRI ( ?arg ) )`

  Here, *DATATYPE-IRI* is the IRI identifying a datatype. For instance, we use `xs:string(?V)` for casting to `xs:string`. Parties defining their own datatypes to be used in RIF exchanged rules may define their own casting function for these datatypes. Labels used for such additional guard

predicates for datatypes not mentioned in the present document MAY follow the same naming convention using the IRI identifying a datatype as function name for the casting function.

- *Domain*:

  The domain for casting functions to XML schema datatypes depends on where the casting is defined according to Section 17.1 of [XPath-Functions]: for all the casting functions to XML schema datatypes the domain of `?arg` is at most the set of all data values in the value spaces of XML schema datatypes such that the conversion to *DATATYPE-IRI* does not raise a type error or an invalid value for cast/constructor error [err:FORG0001] or an invalid lexical value error [err:FOCA0002] according to Section 17.1 of [XPath-Functions]. We will mention additional constraints on the domain for casts to specific datatypes below separately.

- *Mapping:* The mappings for casting functions to XML schema datatypes are defined as follows:

  $I_{external}$(?arg; *DATATYPE-IRI* (`?arg`))(*SV*) = *TV*, which is a value the value space of the datatype with IRI *DATATYPE-IRI* in derived from a type of *SV*, as defined in Section 17.1 of [XPath-Functions].

  If the argument value is outside of its domain, the value of the function is left unspecified. We will mention additional constraints on the mappings for casts to specific datatypes below separately.

Accordingly, the following schemas are defined:

- `( ?arg; xs:anyURI ( ?arg ) )`

  Additional restriction on the Domain: Note that unlike [XPath-Functions] the extent to which an implementation validates the lexical form of `xs:anyURI` is not implementation dependent, but RIF requires all lexical forms of `xs:anyURI` appearing as constants in the `xs:string` symbol space to be castable to `xs:anyURI`.

- `( ?arg; xs:base64Binary( ?arg ) )`
- `( ?arg; xs:boolean( ?arg ) )`
- `( ?arg; xs:date ( ?arg ) )`

  Additional restriction on the Domain: The domain where this function is specified in RIF is further restricted to data values in the value spaces of XML schema datatypes such that the conversion to `xs:date` does not result in a value from the `xs:date` value space outside what [http://www.w3.org/TR/xmlschema11-2/#dt-minimally-conforming minimal conformance] as defined in Section 5.4 of [XML-SCHEMA2] requires for `xs:date`.

- `( ?arg; xs:dateTime ( ?arg ) )`

  Additional restriction on the Domain: The domain where this functions is specified in RIF is further restricted to data values in the value spaces of XML schema datatypes such that the conversion to `xs:date` does not result in a value from the `xs:dateTime` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for `xs:dateTime`.

- `( ?arg; xs:dateTimeStamp ( ?arg ) )`

  Additional restriction on the Domain: Since `xs:dateTimeStamp` is a derived type of `dateTime` the domain if this function is the same as for casting to `xs:dateTime` with the additional restriction that casting to `xs:dateTimeStamp` is only defined for values such that the conversion to `xs:dateTime` has a non-empty timezone component.

- `( ?arg; xs:double ( ?arg ) )`
- `( ?arg; xs:float ( ?arg ) )`
- `( ?arg; xs:hexBinary ( ?arg ) )`
- `( ?arg; xs:decimal ( ?arg ) )`

  Additional restriction on the Domain: The domain where this functions is specified in RIF is further restricted to data values in the value spaces of XML schema datatypes such that the conversion to `xs:decimal` does not result in a value from the `xs:decimal` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for `xs:decimal`.

- `( ?arg; xs:integer( ?arg ) )`

  Additional restriction on the Domain: The domain where this functions is specified in RIF is further restricted to data values in the value spaces of XML schema datatypes such that the conversion to `xs:integer` does not result in a value from the `xs:integer` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for `xs:integer`.

- `( ?arg; xs:long ?arg ) )`
- `( ?arg; xs:int( ?arg ) )`
- `( ?arg; xs:short( ?arg ) )`
- `( ?arg; xs:byte( ?arg ) )`
- `( ?arg; xs:nonNegativeInteger( ?arg ) )`
- `( ?arg; xs:positiveInteger( ?arg ) )`
- `( ?arg; xs:unsignedLong( ?arg ) )`
- `( ?arg; xs:unsignedInt( ?arg ) )`
- `( ?arg; xs:unsignedShort( ?arg ) )`
- `( ?arg; xs:unsignedByte( ?arg ) )`
- `( ?arg; xs:nonPositiveInteger( ?arg ) )`

- `( ?arg; xs:negativeInteger( ?arg ) )`
- `( ?arg; xs:string ( ?arg ) )`

  Additional restrictions on the Domain:

  1. Note that conversions from `xs:float` and `xs:double` to `xs:string` according to Section 17.1.1 of [XPath-Functions] may vary between implementations. Thus, the domain where this functions is specified in RIF is further restricted for data values in the value spaces of XML schema datatypes such that the conversion to `xs:string` is non-ambiguous in a minimally conformant implementation as defined in Section 5.4 of [XML-SCHEMA2].
  2. RIF additionally includes values in the `rdf:XMLLiteral` value space to the domain.

  Additional remark on the mapping:
  If *SV* is a value in the value space of `rdf:XMLLiteral`, then $I_{external}$(?arg; `xs:string`(`?arg`))(*SV*) = *TV* such that *TV* is the string in the lexical space of rdf:XMLLiteral correspoding to *SV* (cf. [RDF-CONCEPTS]).

- `( ?arg; xs:normalizedString ( ?arg ) )`
- `( ?arg; xs:token ( ?arg ) )`
- `( ?arg; xs:language ( ?arg ) )`
- `( ?arg; xs:Name ( ?arg ) )`
- `( ?arg; xs:NCName ( ?arg ) )`
- `( ?arg; xs:NMTOKEN ( ?arg ) )`
- `( ?arg; xs:time ( ?arg ) )`
- `( ?arg; xs:dayTimeDuration ( ?arg ) )`
- `( ?arg; xs:yearMonthDuration ( ?arg ) )`

4.4.2 Casting to `rdf:XMLLiteral`

- *Schema*:

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

- *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:*

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

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

4.4.3 Casting to `rdf:PlainLiteral`

- *Schema*:

  ```
  ( ?arg; rdf:PlainLiteral ( ?arg ) )
  ```

- *Domain*:

  The union of the value spaces of XML schema datatypes.

- *Mapping:*

  Since the value space of `xs:string` is included in the value space of `rdf:PlainLiteral`, the mapping is defined in precisely the same way as for casts to `xs:string`.

4.4.4 `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:*

  $I_{external}$(?arg$_1$?arg$_2$; pred:iri-string (?arg$_1$?arg$_2$))(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
```

4.5 Numeric Functions and Predicates

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

4.5.1 Numeric Functions

The following numeric binary 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].

- *Schema*:

  The schemas for these functions have the general form

  `(?arg1 ?arg2; func:numeric-`*`BINOP`*`(?arg1 ?arg2))`

- *Domains*:

  The domain of these functions is made up of pairs of values from value spaces of `xs:integer`, `xs:double`, `xs:float`, or `xs:decimal` for both arguments such that `op:numeric-`*`BINOP`* as defined in [XPath-

Functions] after type promotion does not result in a numeric operation overflow/underflow error err:FOAR0002, division by zero error err:FOAR0001, or a value from the `xs:decimal` value spaces expressible with sixteen total digits, i.e., RIF requires minimal conformance as defined in Section 5.4 of [XML-SCHEMA2].

- *Mapping:*

  $I_{external}$(?arg_1 ?arg_2; func:numeric-*BINOP*(?arg_1 ?arg_2))(a_1 a_2) = *res* such that *res* is the result of `op:numeric-`*`BINOP`*(a_1', a_2') as defined in [XPath-Functions], in case both a_1 and a_2 belong to their domains. Here, a_1' and a_2' are obtained from a_1 and a_2 as follows:

  - if a_1 and a_2 are both in the value space of `xs:decimal`, in the value space of `xs:float`, or in the value space of `xs:double`, a_1' = a_1 and a_2' = a_2, else
  - if neither a_1 nor a_2 is in the value space of `xs:double`, a_1' and a_2' are obtained by promoting a_1 and a_2 to `xs:float`, as defined in Appendix B.1 of [XPath], else
  - a_1' and a_2' are obtained by promoting a_1 and a_2 to `xs:double`, as defined in Appendix B.1 of [XPath].

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

Accordingly, the following schemas are defined:

- (?arg_1 ?arg_2; `func:numeric-add(` ?arg_1 ?arg_2)) (adapted from `op:numeric-add`)
- (?arg_1 ?arg_2; `func:numeric-subtract(` ?arg_1 ?arg_2)) (adapted from `op:numeric-subtract`)
- (?arg_1 ?arg_2; `func:numeric-multiply(` ?arg_1 ?arg_2)) (adapted from `op:numeric-multiply`)
- (?arg_1 ?arg_2; `func:numeric-divide(` ?arg_1 ?arg_2)) (adapted from `op:numeric-divide`)
- (?arg_1 ?arg_2; `func:numeric-integer-divide(` ?arg_1 ?arg_2)) (adapted from `op:numeric-integer-divide`)
- (?arg_1 ?arg_2; `func:numeric-mod(` ?arg_1 ?arg_2)) (adapted from `op:numeric-integer-mod`)

4.5.2 Numeric Predicates

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both a_1 and a_2 belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:numeric-equal(?arg1 ?arg2))(a_1 a_2) = **t** if and only if op:numeric-equal(a_1, a_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both a_1 and a_2 belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:numeric-less-than(?arg1 ?arg2))(a_1 a_2) = **t** if and only if op:numeric-less-than(a_1, a_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

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

4.5.2.4 `pred:numeric-not-equal`

- *Schema*:

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

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

4.5.2.5 `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 and false otherwise.

4.5.2.6 `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 and false otherwise.

4.6 Functions and Predicates on Boolean Values

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

4.6.1 Functions on Boolean Values

4.6.1.1 `func:not` (adapted from `fn:not`)

- *Schema*:

  `(?arg ; func:not( ?arg ) )`

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg ; func:numeric-mod(?arg))(a_1) = *res* such that *res* is the result of fn:not(a_1) as defined in [XPath-Functions], in case a_1 belongs to its domain.

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

4.6.2 Predicates on Boolean Values

4.6.2.1 `pred:boolean-equal` (adapted from `op:boolean-equal`)

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both a_1 and a_2 belong to their domains, I_{truth} ∘ $I_{external}$(?arg1 ?arg2; pred:boolean-equal(?arg1 ?arg2))(a_1 a_2) = **t** if and only if op:boolean-equal(a_1, a_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

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

4.6.2.2 `pred:boolean-less-than` (adapted from `op:boolean-less-than`)

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both a_1 and a_2 belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:boolean-less-than(?arg1 ?arg2))(a_1 a_2) = **t** if and only if op:boolean-less-than(a_1, a_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

4.6.2.3 `pred:boolean-greater-than` (adapted from `op:boolean-greater-than`)

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both a_1 and a_2 belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:boolean-greater-than(?arg1 ?arg2))(a_1 a_2) = **t** if and only if op:boolean-greater-than(a_1, a_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

4.7 Functions and Predicates on Strings

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

4.7.1 Functions on Strings

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for $\texttt{?comparand}_1$ and $\texttt{?comparand}_2$; the domain of `?collation` is empty.

- *Mapping:*

  $I_{external}$(?comparand$_1$?comparand$_2$; func:compare(?comparand$_1$?comparand$_2$))(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 default codepoint collation as defined in Section 7.3.1 of [XPath-Functions]. 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, where the default behavior in RIF is the codepoint collation.

  If an argument value is outside of its domain, the value of the function is left unspecified. Note that specifically the defined domain for the `?collation` argument is empty in RIF. That means RIF does not prescribe any specific collation apart from the default codepoint collation and - consequently - the result of this function with a given collation argument is not defined by RIF and may vary between implementations.

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

- *Schemata*:

`( ?arg1 ?arg2; func:concat(?arg1 ?arg2 ) )`

`...`

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

- *Domains*:

  Following the definition of fn:concat this function casts its arguments to `xs:string`. Thus, the domain for all arguments is the union of all values castable to `xs:string` as defined in Section Datatype Conversion and Casting above.

- *Mapping:*

  $I_{external}$(?arg1 ... ?arg*n*; func:concat(?arg1 ... ?arg*n*))(s1 ... s*n*) = *res* such that *res* is the result of `fn:concat`(s1 ... s*n*) 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.

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

- *Schemata*:

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

  `( ?arg1 ?arg2 ?arg3; func:string-join(?arg1 ?arg2 ?arg3 ) )`

  `...`

  `( ?arg1 ?arg2 ... ?argn-1 ?argn; func:string-join(?arg1 ?arg2 ... ?argn-1 ?argn ) )`

- *Domains*:

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

- *Mapping:*

  $I_{external}$(?arg1 ... ?arg*n-1* ?arg*n*; func:string-join(?arg1 ... ?arg*n-1* ?arg*n*))(s1 ... s*n-1* s*n*) = *res* such that *res* is the result of `fn:string-join`((s1 ... s*n-1*) s*n*) 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.

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

- *Schemata*:

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

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

- *Domains*:

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

- *Mapping:*

  $I_{external}$(?sourceString ?startingLoc ?length; func:substring(?sourceString ?startingLoc ?length))(src loc len) = *res* such that *res* is the result of `fn:substring`(src loc len) 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.

Note that, as in XPath-Functions, the first character of a string is located at position 1, not position 0.

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg; func:string-length(?arg))(s) = *res* such that *res* is the result of `fn:string-length`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg; func:upper-case(?arg))(s) = *res* such that *res* is the result of `fn:upper-case`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg; func:lower-case(?arg))(s) = *res* such that *res* is the result of `fn:lower-case`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

4.7.1.8 `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:*

  $I_{external}$(?arg; func:encode-for-uri(?arg))(s) = *res* such that *res* is the result of `fn:encode-for-uri`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg; func:iri-to-uri(?arg))(s) = *res* such that *res* is the result of `fn:iri-to-uri`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg; func:escape-html-uri(?arg))(s) = *res* such that *res* is the result of `fn:escape-html-uri`(s) as defined in [XPath-Functions], in case the argument belongs to its domain.

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

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for $?arg_1$ and $?arg_2$; the domain of `?collation` is empty.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:substring-before(?arg1 ?arg2))(s_1 s_2) = *res*, such that *res* is the substring of s_1 that precedes in the value of s_1 the first occurrence of a sequence of collation units that provides a minimal match to the collation units of s_2 according to the default codepoint collation as defined in Section 7.3.1 of [XPath-Functions].

  If any argument value is outside of its domain, the value of the function is left unspecified. Note that specifically the defined domain for the `?collation` argument is empty in RIF. That means RIF does not prescribe any specific collation apart from the default codepoint collation and - consequently - the result of this function with a given collation argument is not defined by RIF and may vary between implementations.

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for $?arg_1$ and $?arg_2$; the domain of `?collation` is empty.

- *Mapping:*

$I_{external}$(?arg_1 ?arg_2; func:substring-after(?arg_1 ?arg_2))(s_1 s_2) = *res*, such that *res* is the substring of s_1 that follows in the value of ?s_1 the first occurrence of a sequence of collation units that provides a minimal match to the collation units of s_2 according to the default codepoint collation as defined in Section 7.3.1 of [XPath-Functions].

If any argument value is outside of its domain, the value of the function is left unspecified. Note that specifically the defined domain for the `?collation` argument is empty in RIF. That means RIF does not prescribe any specific collation apart from the default codepoint collation and - consequently - the result of this function with a given collation argument is not defined by RIF and may vary between implementations.

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for the first three arguments and all values in the value space of `xs:string` that are valid flags following Section 7.6.1.1 of [XPath-Functions] for `?flags`.

- *Mapping:*

  $I_{external}$ (?input ?pattern ?replacement ?flags; func:replace(?input ?pattern ?replacement ?flags)) (i p r f) = *res*, such that *res* is the result of `fn:replace`(i p r f) as defined in [XPath-Functions], in case the arguments belongs to their domains.

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

4.7.2 Predicates on Strings

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for $?arg_1$ and $?a_2$; the domain of `?collation` is empty.

- *Mapping:*

  When all arguments belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:contains(?arg1 ?arg2))(s_1 s_2) = **t** if and only if fn:contains(s_1, s_2) returns `true`, as defined in [XPath-Functions], **f** otherwise. 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 default codepoint collation as defined in Section 7.3.1 of [XPath-Functions].

  If an argument value is outside of its domain, the truth value of the function is left unspecified. Note that specifically the defined domain for the `?collation` argument is empty in RIF. That means RIF does not prescribe any specific collation apart from the default codepoint collation and - consequently - the result of this function with a given collation argument is not defined by RIF and may vary between implementations.

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for $?arg_1$ and $?a_2$; the domain of `?collation` is empty.

- *Mapping:*

  When all arguments belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:starts-with(?arg1 ?arg2))(s_1 s_2) = **t** if and only if fn:starts-with(s_1, s_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for `?arg1` and `?a2`; the domain of `?collation` is empty.

- *Mapping:*

  When all arguments belong to their domains, $I_{truth} \circ I_{external}$(?arg1 ?arg2; pred:ends-with(?arg1 ?arg2))(s_1 s_2) = **t** if and only if fn:ends-with(s_1, s_2) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

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

- *Schema*:

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

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

- *Domains*:

  The value space of `xs:string` for the first two arguments and all values in the value space of `xs:string` that are valid flags following Section 7.6.1.1 of [XPath-Functions] for `?flags`.

- *Mapping:*

  When all arguments belong to their domains, $I_{truth} \circ I_{external}$(?input ?pattern ?flags; pred:matches(?input ?pattern ?flags))(i p

f) = **t** if and only if pred:matches(i p f) returns `true`, as defined in [XPath-Functions], **f** otherwise.

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

4.8 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.

4.8.1 Functions on Dates, Times, and Durations

4.8.1.1 `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 the 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." The RIF version of func:year-from-dateTime does not support "empty sequences".

4.8.1.2 `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:*

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

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

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

4.8.1.3 `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:*

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

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

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

4.8.1.4 `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:*

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

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

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

4.8.1.5 `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:*

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

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

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

4.8.1.6 `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:*

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

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

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

4.8.1.7 `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:*

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

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

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

4.8.1.8 `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:*

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

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

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

4.8.1.9 `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:*

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

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

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

4.8.1.10 `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:*

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

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

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

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

- *Schema*:

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

- *Mapping:*

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

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

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

4.8.1.12 `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:*

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

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

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

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

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

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

- *Schema*:

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

- *Domain*:

  The value space of `xs:dateTimeStamp`.

- *Mapping:*

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

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

  If the argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:dateTimeStamp` instead of `xs:dateTime`, i.e. RIF leaves the return value for `xs:dateTime` values without a timezone unspecified.

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

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

- *Schema*:

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

- *Domain*:

  The values of value space `xs:date` with a timezone component.

- *Mapping:*

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

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

  If the argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:date` values with a timezone component, i.e. RIF leaves the return value for `xs:date` values without a timezone unspecified.

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

- *Schema*:

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

- *Domain*:

  The values of value space `xs:time` with a timezone component.

- *Mapping:*

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

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

  If the argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:time` values with a timezone component, i.e. RIF leaves the return value for `xs:time` values without a timezone unspecified.

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

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:subtract-dateTimes(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:subtract-dateTimes(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:dateTimeStamp`s instead of `xs:dateTime`, i.e. RIF leaves the return value for `xs:dateTime` arguments values without a timezone unspecified.

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

- *Schema*:

  `( ?arg`$_1$ `?arg`$_2$`; func:subtract-dates( ?arg`$_1$ `?arg`$_2$ `) )`

- *Domain*:

  The value space of `xs:date` with given timezone for both arguments.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:subtract-dates(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:subtract-dates(s$_1$ s$_2$) as defined in [XPath-Functions].

  If any argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:date`s with explicit timezone, i.e. RIF leaves the return value for `xs:date` arguments values without a timezone unspecified.

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

- *Schema*:

  `( ?arg`$_1$ `?arg`$_2$`; func:subtract-times( ?arg`$_1$ `?arg`$_2$ `) )`

- *Domain*:

  The value space of `xs:time` with given timezone for both arguments.

- *Mapping:*

  $I_{external}$(?arg_1 ?arg_2; func:subtract-times(?arg_1 ?arg_2))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-times(s_1 s_2) as defined in [XPath-Functions].

  If any argument value is outside of its domain, the value of the function is left unspecified. Note that RIF restricts the domain of this function to `xs:time`s with explicit timezone, i.e. RIF leaves the return value for `xs:date` arguments values without a timezone unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:yearMonthDuration` for both ?arg_1 and ?arg_2 such that fn:add-yearMonthDurations does not result in a duration operation overflow/underflow error err:FODT0002 or in a value from the `xs:yearMonthDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg_1 ?arg_2; func:add-yearMonthDurations(?arg_1 ?arg_2))(s_1 s_2) = *res*

  such that *res* is the result of fn:add-yearMonthDurations(s_1 s_2) as defined in [XPath-Functions].

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

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:yearMonthDuration` for both $\texttt{?arg}_1$ and $\texttt{?arg}_2$ such that fn:subtract-yearMonthDurations does not result in a duration operation overflow/underflow error err:FODT0002 or in a value from the `xs:yearMonthDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$($\texttt{?arg}_1$ $\texttt{?arg}_2$; func:subtract-yearMonthDurations($\texttt{?arg}_1$ $\texttt{?arg}_2$))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-yearMonthDurations(s_1 s_2) as defined in [XPath-Functions].

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

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:yearMonthDuration` for $\texttt{?arg}_1$ and `xs:integer`, `xs:double`, `xs:float`, or `xs:decimal` for $\texttt{?arg}_2$ such that fn:multiply-yearMonthDuration does not result in a duration operation overflow/underflow error err:FODT0002, NaN supplied as double value error err:FOCA0005, or in a value from the `xs:yearMonthDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$($\texttt{?arg}_1$ $\texttt{?arg}_2$; func:multiply-yearMonthDuration($\texttt{?arg}_1$ $\texttt{?arg}_2$))(s_1 s_2) = *res*

  such that *res* is the result of fn:multiply-yearMonthDuration(s_1 s_2) as defined in [XPath-Functions].

If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:yearMonthDuration` for $?arg_1$ and `xs:integer`, `xs:double`, `xs:float`, or `xs:decimal` for $?arg_2$ such that fn:divide-yearMonthDuration does not result in a duration operation overflow/underflow error err:FODT0002, NaN supplied as double value error err:FOCA0005, or in a value from the `xs:yearMonthDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:divide-yearMonthDuration(?arg1 ?arg2))(s_1 s_2) = *res*

  such that *res* is the result of fn:divide-yearMonthDuration(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:yearMonthDuration` for both $?arg_1$ and $?arg_2$ such that fn:divide-yearMonthDuration-by-yearMonthDuration does not result in a duration operation overflow/underflow error err:FODT0002 or in a value

from the `xs:yearMonthDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:divide-yearMonthDuration-by-yearMonthDuration(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:divide-yearMonthDuration-by-yearMonthDuration(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value space of `xs:dayTimeDuration` for ?arg$_1$ and ?arg$_2$ such that fn:add-dayTimeDurations does not result in a duration operation overflow/underflow error err:FODT0002 or in a value from the `xs:dayTimeDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:add-dayTimeDurations(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:add-dayTimeDurations(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value space of `xs:dayTimeDuration` for $?arg_1$ and $?arg_2$ such that fn:subtract-dayTimeDurations does not result in a duration operation overflow/underflow error err:FODT0002 or in a value from the `xs:dayTimeDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:subtract-dayTimeDurations(?arg1 ?arg2))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-dayTimeDurations(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dayTimeDuration` for $?arg_1$ and `xs:integer`, `xs:double`, `xs:float`, or `xs:decimal` for $?arg_2$ such that fn:multiply-dayTimeDuration does not result in a duration operation overflow/ underflow error err:FODT0002, NaN supplied as double value error err:FOCA0005, or in a value from the `xs:dayTimeDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:multiply-dayTimeDuration(?arg1 ?arg2))(s_1 s_2) = *res*

such that *res* is the result of fn:multiply-dayTimeDuration(s_1 s_2) as defined in [XPath-Functions].

If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dayTimeDuration` for `?arg1` and `xs:integer`, `xs:double`, `xs:float`, or `xs:decimal` for `?arg2` such that fn:divide-dayTimeDuration does not result in a duration operation overflow/ underflow error err:FODT0002, NaN supplied as double value error err:FOCA0005, or in a value from the `xs:dayTimeDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:divide-dayTimeDuration(?arg1 ?arg2))(s_1 s_2) = *res*

  such that *res* is the result of fn:divide-dayTimeDuration(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

The domain of this functions is made up of pairs of values from value spaces of `xs:dayTimeDuration` for both $?arg_1$ and $?arg_2$ such that fn:divide-dayTimeDuration-by-dayTimeDuration does not result in a duration operation overflow/underflow error err:FODT0002 or in a value from the `xs:dayTimeDuration` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:divide-dayTimeDuration-by-dayTimeDuration(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:divide-dayTimeDuration-by-dayTimeDuration(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dateTime` for $?arg_1$ and `xs:yearMonthDuration` for $?arg_2$ such that fn:add-yearMonthDuration-to-dateTime does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:dateTime` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:add-yearMonthDuration-to-dateTime(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:add-yearMonthDuration-to-dateTime(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:date` for `?arg1` and `xs:yearMonthDuration` for `?arg2` such that fn:add-yearMonthDuration-to-date does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:date` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:add-yearMonthDuration-to-date(?arg1 ?arg2))(s_1 s_2) = *res*

  such that *res* is the result of fn:add-yearMonthDuration-to-date(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dateTime` for `?arg1` and `xs:dayTimeDuration` for `?arg2` such that fn:add-dayTimehDuration-to-dateTime does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:dateTime` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

$I_{external}$(?arg_1 ?arg_2; func:add-dayTimeDuration-to-dateTime(?arg_1 ?arg_2))(s_1 s_2) = *res*

such that *res* is the result of fn:add-dayTimeDuration-to-dateTime(s_1 s_2) as defined in [XPath-Functions].

If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:date` for `?arg1` and `xs:dayTimeDuration` for `?arg2` such that fn:add-dayTimehDuration-to-date does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:date` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg_1 ?arg_2; func:add-dayTimeDuration-to-date(?arg_1 ?arg_2))(s_1 s_2) = *res*

  such that *res* is the result of fn:add-dayTimeDuration-to-date(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

The domain of this functions is made up of pairs of values from value spaces of `xs:time` for $?arg_1$ and `xs:dayTimeDuration` for $?arg_2$ such that fn:add-dayTimehDuration-to-time does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:time` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$($?arg_1$ $?arg_2$; func:add-dayTimeDuration-to-time($?arg_1$ $?arg_2$))(s_1 s_2) = *res*

  such that *res* is the result of fn:add-dayTimeDuration-to-time(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dateTime` for $?arg_1$ and `xs:yearMonthDuration` for $?arg_2$ such that fn:subtract-yearMonthDuration-from-dateTime does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:dateTime` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$($?arg_1$ $?arg_2$; func:subtract-yearMonthDuration-from-dateTime($?arg_1$ $?arg_2$))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-yearMonthDuration-from-dateTime(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:date` for `?arg1` and `xs:yearMonthDuration` for `?arg2` such that fn:subtract-yearMonthDuration-from-date does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:date` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg1 ?arg2; func:subtract-yearMonthDuration-from-date(?arg1 ?arg2))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-yearMonthDuration-from-date(s_1 s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

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

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:dateTime` for `?arg1` and `xs:dayTimeDuration` for `?arg2` such that fn:subtract-dayTimehDuration-from-dateTime does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:dateTime` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

$I_{external}$(?arg$_1$?arg$_2$; func:subtract-dayTimeDuration-from-dateTime(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

such that *res* is the result of fn:subtract-dayTimeDuration-from-dateTime(s$_1$ s$_2$) as defined in [XPath-Functions].

If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

  `( ?arg`$_1$ `?arg`$_2$ `; func:subtract-dayTimeDuration-from-date( ?arg`$_1$ `?arg`$_2$ `) )`

- *Domain*:

  The domain of this functions is made up of pairs of values from value spaces of `xs:date` for `?arg`$_1$ and `xs:dayTimeDuration` for `?arg`$_2$ such that fn:subtract-dayTimehDuration-from-date does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:date` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$(?arg$_1$?arg$_2$; func:subtract-dayTimeDuration-from-date(?arg$_1$?arg$_2$))(s$_1$ s$_2$) = *res*

  such that *res* is the result of fn:subtract-dayTimeDuration-from-date(s$_1$ s$_2$) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

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

- *Schema*:

  `( ?arg`$_1$ `?arg`$_2$ `; func:subtract-dayTimeDuration-from-time( ?arg`$_1$ `?arg`$_2$ `) )`

- *Domain*:

The domain of this functions is made up of pairs of values from value spaces of `xs:time` for $?arg_1$ and `xs:dayTimeDuration` for $?arg_2$ such that fn:subtract-dayTimehDuration-from-time does not result in a date/time operation overflow/underflow error err:FODT0001 or in a value from the `xs:time` value space outside what minimal conformance as defined in Section 5.4 of [XML-SCHEMA2] requires for durations.

- *Mapping:*

  $I_{external}$($?arg_1$ $?arg_2$; func:subtract-dayTimeDuration-from-time($?arg_1$ $?arg_2$))(s_1 s_2) = *res*

  such that *res* is the result of fn:subtract-dayTimeDuration-from-Time$_1$ s_2) as defined in [XPath-Functions].

  If the arguments are outside of the domain, the value of the function is left unspecified.

4.8.2 Predicates on Dates, Times, and Durations

4.8.2.1 `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} ∘ $I_{external}$($?arg_1$ $?arg_2$; pred:dateTime-equal($?arg_1$ $?arg_2$))(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.

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

- *Schema*:

  ```
  ( ?arg1 ?arg2; pred:dateTime-less-than(?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} ○ $I_{external}$(?arg$_1$?arg$_2$; pred:dateTime-less-than(?arg$_1$?arg$_2$))(s_1 s_2) = **t**

  if and only if op:dateTime-less-than(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.

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

- *Schema*:

  ```
  ( ?arg1 ?arg2; pred:dateTime-greater-than(?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} ○ $I_{external}$(?arg$_1$?arg$_2$; pred:dateTime-greater-than(?arg$_1$?arg$_2$))(s_1 s_2) = **t**

  if and only if op:dateTime-greater-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

  if and only if op:date-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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:date-less-than(?arg1 ?arg2))(s_1 s_2) = **t**

  if and only if op:date-less-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:date-greater-than(?arg1 ?arg2))(s_1 s_2) = **t**

if and only if op:date-greater-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

  if and only if op:time-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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

  if and only if op:time-less-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:time-greater-than(?arg1 ?arg2))(s_1 s_2) = **t**

  if and only if op:time-greater-than(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.

4.8.2.10 `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.

- *Mapping:*

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

  if and only if op:duration-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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:dayTimeDuration-less-than(?arg1 ?arg2))(s_1 s_2) = **t**

  if and only if op:dayTimeDuration-less-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

  When both s_1 and s_2 belong to their domains, I_{truth} ○ $I_{external}$(?arg1 ?arg2; pred:dayTimeDuration-greater-than(?arg1 ?arg2))(s_1 s_2) = **t**

  if and only if op:dayTimeDuration-greater-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

  if and only if op:yearMonthDuration-less-than(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.

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

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

  if and only if op:yearMonthDuration-greater-than(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.

4.8.2.15 `pred:dateTime-not-equal`

- *Schema*:

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

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

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

- *Schema*:

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

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.

4.8.2.17 `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.

4.8.2.18 `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.

4.8.2.19 `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.

4.8.2.20 `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.

4.8.2.21 `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.

4.8.2.22 `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.

4.8.2.23 `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.

4.8.2.24 `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.

4.8.2.25 `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.

4.8.2.26 `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.

4.8.2.27 `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.

4.8.2.28 `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.

4.9 Functions and Predicates on rdf:XMLLiterals

4.9.1 `pred:XMLLiteral-equal`

- *Schema*:

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

- *Domains*:

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

- *Mapping:*

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

4.9.2 `pred:XMLLiteral-not-equal`

- *Schema*:

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

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

4.10 Functions and Predicates on rdf:PlainLiteral

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

Editor's Note: Issues which are still open in the rdf:PlainLiteral specification might imply future changes on the functions and predicates defined here. For instance, `plfn:compare` and `plfn:length` are curently marked AT RISK. We could subsume these functions under a single func:compare and func:compare function, instead of defining separate functions for `xs:string` and `rdf:PlainLiteral`, or drop them alltogether for redundancy.

4.10.1 Functions on rdf:PlainLiteral

4.10.1.1 `func:PlainLiteral-from-string-lang` (adapted from `plfn:PlainLiteral-from-string-lang`)

- *Schema*:

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

- *Domains*:

  The value space of `xs:string` for $?arg_1$ and the intersection of the elements of the value space of `xs:string` which represent valid language tags according to [BCP-47] for $?arg_2$.

- *Mapping:*

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

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

4.10.1.2 `func:string-from-PlainLiteral` (adapted from `plfn:string-from-PlainLiteral`)

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg ; func:string-from-PlainLiteral(?arg))(t) = *res* such that *res* is the string part s of t if t is a pair < s, l > or *res* = t if t is a string value.

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

4.10.1.3 `func:lang-from-PlainLiteral` (adapted from `plfn:lang-from-PlainLiteral`)

- *Schema*:

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

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg ; func:lang-from-PlainLiteral(?arg))(t) = *l* such that *l* is the language tag string of t if t is a pair < s, l > and `""^^xs:string` if t is a string value.

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

4.10.1.4 `func:PlainLiteral-compare` (adapted from `plfn:compare`)

- *Schema*:

  ```
  ( ?comparand1 ?comparand2; func:PlainLiteral-
  compare(?comparand1 ?comparand2) )

  ( ?comparand1 ?comparand2 ?collation;
  func:PlainLiteral-
  compare(?comparand1 ?comparand2 ?collation) )
  ```

- *Domains*:

The value space of `rdf:PlainLiteral` for $?comparand_1$ and $\$comparand_2$, and the empty set for `?collation`.

- *Mapping:*

  $I_{external}$(?comparand$_1$?comparand$_2$; func:PlainLiteral-compare(?comparand$_1$?comparand$_2$))(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:PlainLiteral`, or two string values t_1=s_1 and t_2=s_2, respectively, then *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 default codepoint collation as defined in Section 7.3.1 of [XPath-Functions].

  In case an argument value is outside of its domain, or if the language tags of the values for $?comparand_1$ and $\$comparand_2$ differ, the function value is left unspecified. That means RIF does not prescribe any specific collation apart from the default codepoint collation and - consequently - the result of this function with a given collation argument is not defined by RIF and may vary between implementations.

4.10.1.5 `func:PlainLiteral-length` (adapted from `plfn:length`)

- *Schema*:

  ```
  ( ?arg ; func:PlainLiteral-length( ?arg ) )
  ```

- *Domain*:

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

- *Mapping:*

  $I_{external}$(?arg ; func:PlainLiteral-length(?arg))(s) = *res* such that *res* is a value in the value space of `xs:integer` equal to the length in characters of the string part *s* of the argument if it is a pair (*s*,l), or the argument is a string value *s*, respectively.

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

4.10.2 Predicates on rdf:PlainLiteral

4.10.2.1 `pred:matches-language-range` (adapted from `plfn:matches-language-range`)

- *Schema*:

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

- *Domains*:

  The value space of `rdf:PlainLiteral` for `?input` and the values of value space `xs:string` that correspond to valid language tags according to [BCP-47] for `?range`.

- *Mapping:*

  Whenever both arguments are within their domains, <p>$I_{external}$ (?input ?range; pred:matches-language-range(?input ?range))(i r) = **t** if and only if `plfn:matches-language-range`(i r) as specified in [RDF-PLAINLITERAL] returns `true`, **f** otherwise.

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

4.11 Functions and Predicates on RIF Lists

RIF Lists are similar to list and array types in many systems, as well as XPath/ XQuery Sequences [XPath-Functions]. They differ from XPath as follows:

- They are called "lists" instead of sequences (so the "subsequence" function is called "sublist")
- Positions (indexes) count from zero, instead of one, and negative indexes are defined to count back from the end of the list
- They are not limited to containing only atomic data; in particular, they may contain other lists.
- There is no equivalence between an atomic value and a singleton list; in RIF these are distinct values.

4.11.1 Position Numbering

The positions in a list are numbered starting with zero. That is, in a list of length n+1, the first item has position 0, and the last item has position n. When a negative position number is provided to a builtin, the length of the list is added to it before it is used, so it effectively counts backward from the end of the list: position -1 points to the last item in the list, i.e. corresponds effectively to position n, etc.

Feature At Risk #5: Indexing from zero

Note: This feature is "at risk" and may be altered or removed from this specification based on feedback. If you have concerns about this or information which may be useful in our eventual decision, please tell us at public-rif-comments@w3.org. See the full list of features "at risk" in RIF.

The list positions are here defined to start from zero, while in XPath-Functions, the sequence positions and string positions (as seen in func:substring) start from one. We may address this source of confusion by changing the definitions of functions and predicates which use position numbering.

4.11.2 Item Comparison

List items are compared for equality (as required by many of these builtins) using normal RIF equality testing, not datatype equality (e.g., pred:numeric-equal).

Several list builtins need to establish inequality in order to compute a result. If all the compared items are literals or lists, this is not a problem, but if they are rif:local or rif:iri terms, the knowledge base is unlikely to contain inequality information. This may lead to counter-intuitive results. For example, the empty ruleset does not entail `External(func:index-of( List(ex:foo ex:bar) ex:foo) ) == List(0)`, because the empty ruleset provides no indication whether `eg:foo = eg:bar`.

4.11.3 Predicates on RIF Lists

4.11.3.1 `pred:is-list`

- *Schema*:

  `(?object; pred:is-list(?object))`

- *Domains*:

  ?object: unrestricted

- *Mapping:*

  $I_{truth} \circ I_{external}$(?object; pred:is-list (?object))(s) = **t** if and only if there exists some (t_0, ... , t_n) such that either $I_{list}(t_0 \ldots t_n)$ = s or $I_{tail}(t_0, \ldots, t_n)$ = s, and **f** otherwise.

  Note, that since the syntactic forms of open and closed lists using the `List` operator refer to I_{list}^{-1} and I_{tail}^{-1}, respectively, `pred:is-list` is always true on these syntactic forms. Further note that per definition $I_{list}(D_{ind})$ is disjoint from the value spaces of all data types in ***DTS***, and

consequently `pred:is-list` is always false on constants belonging to a datatype supported by the RIF dialect at hand. this is illustrated by the following examples.

- *Examples*
  - `External(pred:is-list(List(0 1 2 3)))` will evaluate to **t** in any interpretation.
  - `External(pred:is-list(1))` will evaluate to **f** in any interpretation.
  - `External(pred:is-list(List(0 1 2 List(3 4))))` will evaluate to **t** in any interpretation.
  - `External(pred:is-list(List(0 1 2 | List(3 4))))` will evaluate to **t** in any interpretation.
  - `External(pred:is-list(List(1 | 2)))` will evaluate to **t** in any interpretation.

4.11.3.2 `pred:list-contains`

- *Schema*:

```
(?list ?item; pred:list-contains(?list ?item))
```

- *Domains*:

  ?list: D_{list}
  ?item: unrestricted

- *Mapping*

  $I_{truth} \circ I_{external}$(?list ?item; pred:list-contains (?list ?item))(l s) = **t** if and only if $I_{list}^{-1}(l) = (t_0 \ldots t_n)$ such that $t_i = s$ for some i between 0 and n, and **f** otherwise.

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

- *Examples*
  - `External(pred:list-contains(List(0 1 2 3 4) 2)` will evaluate to **t** in any interpretation.
  - `External(pred:list-contains(List(0 1 2 3 4 5 2 2) 2)` will evaluate to **t** in any interpretation.
  - `External(pred:list-contains(List(2 2 3 4 5 2 2) 1)` will evaluate to **f** in any interpretation.
  - `External(pred:list-contains(List() 1)` will evaluate to **f** in any interpretation.
  - `External(pred:list-contains(List(0 1 2 3 List(7 8)) List(7 8))` will evaluate to **t** in any interpretation.

- `External(pred:list-contains(List(0 1 2 3 List(7 8)) List(7 7))` will evaluate to **f** in any interpretation.

4.11.4 Functions on RIF Lists

4.11.4.1 `func:make-list`

- *Schema*:

  `(?item1 ... ?itemn; func:make-list(?item1 ... ?itemn))`

- *Domains*:

  ?item0: unrestricted
  ...
  ?item*n*: unrestricted

- *Mapping*:

  Returns a list of the arguments ?item$_1$, ... ?item$_n$, in the same order they appear as arguments. That is, $I_{external}$(?item$_1$ ... ?item$_n$; func:make-list(?item$_1$ ... ?item$_n$))(s$_1$ ... s$_n$) = I_{list}(s$_1$ ... s$_n$)

- *Note*:

  This function is useful in RIF Core because the List construction operator is syntactically prohibited from being used with variables.

- *Examples*:

```
External( func:make-list(0 1 2) ) = List(0 1 2)
External( func:make-list() ) = List()
External( func:make-list(0) ) = List(0)
External( func:make-list(0 1 List(20 21))) = List(0 1 List(20 21))
External( func:make-list(List(0 1))) = List(List(0 1))
```

4.11.4.2 `func:count` (adapted from `fn:count`)

- *Schema*:

  `(?list; func:count(?list))`

- *Domains*:

  ?list: D_{list}

- *Mapping*:

Returns the number of entries in the list (the length of the list). That is, $I_{external}$(?list; func:count(?list))(I) = n if I_{list}^{-1}(I) = (t_0 ... t_{n-1}) is an element of D_{ind}^n.

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

- *Examples*:

```
External(func:count(List(0 1 2 3 4))) = 5
External(func:count(List(0))) = 1
External(func:count(List(0 0 0))) = 3
External(func:count(List())) = 0
```

4.11.4.3 `func:get`

- *Schema*:

```
(?list ?position; func:get(?list ?position))
```

- *Domains*:

?list: D_{list}
?position: value space of xs:int

- *Mapping*:

Returns the item at the given position in the list. That is, $I_{external}$(?list ?position; func:get(?list ?position))(I i) = t_i if I_{list}^{-1}(I) = (t_0 ... t_n) and i corresponds to a position between 0 and n, as defined in Section Position Numbering.

If an argument value is outside of its domain or i does not correspond to a position between 0 and n as described in Section Position Numbering the value of the function is left unspecified.

- *Examples*:

```
External( func:get(List(0 1 2 3 4) 0) ) = 0
External( func:get(List(0 1 2 3 4) 1) ) = 1
External( func:get(List(0 1 2 3 4) 4) ) = 4
External( func:get(List(0 1 2 3 4) -1) ) = 4
External( func:get(List(0 1 2 3 4) -5) ) = 0
External( func:get(List(0 1 2 3 4) -10) ) = (unspecified)
External( func:get(List(0 1 2 3 4) 5) ) = (unspecified)
```

4.11.4.4 `func:sublist` (adapted from `fn:subsequence`)

- *Schema*:

```
(?list ?start ?stop; func:sublist(?list ?start ?stop))

(?list ?start; func:sublist(?list ?start))
```

- *Domains*:

  ?list: D_{list}
  ?start: value space of xs:int
  ?stop: value space of xs:int

- *Mapping*:

  Returns a list, containing (in order) the items starting at position '?start' and continuing up to, but not including, the '?stop' position, if '?start' is before '?stop'. The '?stop' position may be omitted, in which case it defaults to the length of the list. That is, $I_{external}$(?list ?start ?stop; func:sublist(?list ?start ?stop))(l i j) = $I_{list}(t_i \ldots t_j)$ if $I_{list}^{-1}(l) = (t_0 \ldots t_n)$ such that i and j correspond to positions between 0 and n, as defined in Section Position Numbering. In case i is omitted it defaults to n.

  If an argument value is outside of its domain or, respectively, i and j do not correspond to positions between 0 and n such that the corresponding position of i is smaller than the corresponding position of j, as described in Section Position Numbering the value of the function is left unspecified.

- *Note*:

  This differs from XPath's fn:subsequence function in using a 'stop' position parameter instead of a 'length' parameter (in addition to the name change, the zero-based indexing, and allowing negative indexes).

- *Examples*:

```
External( func:sublist(List(0 1 2 3 4) 0 0) ) = List()
External( func:sublist(List(0 1 2 3 4) 0 1) ) = List(0)
External( func:sublist(List(0 1 2 3 4) 0 4) ) = List(0 1 2 3)
External( func:sublist(List(0 1 2 3 4) 0 5) ) = List(0 1 2 3 4)
External( func:sublist(List(0 1 2 3 4) 0 10) ) = List(0 1 2 3 4)
External( func:sublist(List(0 1 2 3 4) 0 -2) ) = List(0 1 2)
External( func:sublist(List(0 1 2 3 4) 2 4) ) = List(2 3)
External( func:sublist(List(0 1 2 3 4) 2 -2) ) = List(2)
External( func:sublist(List(0 1 2 3 4) 0) ) = List(0 1 2 3 4)
External( func:sublist(List(0 1 2 3 4) 3) ) = List(3 4)
External( func:sublist(List(0 1 2 3 4) -2) ) = List(3 4)
```

4.11.4.5 `func:append`

- *Schema*:

  `(?list ?item1 ... ?itemn; func:append(?list ?item1 ... ?itemn))`

- *Domains*:

  ?list: D_{list}
  ?item$_1$: unrestricted
  ...
  ?item$_n$: unrestricted

- *Mapping*:

  Returns a list consisting of all the items in ?list, followed by ?item$_i$, for each i, 1 <= i <= n. That is, $I_{external}$(?list ?item$_1$ ... ?item$_n$; func:append(?list ?item$_1$ ... ?item$_n$))(l s_1 ... s_n) = I_{list}(t_0 ... t_k s_1 ... s_n) if I_{list}^{-1}(l) = (t_0 ... t_k).

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

- *Examples*:

```
External( func:append(List(0 1 2) 3) )= List(0 1 2 3)
External( func:append(List(0 1 2) 3 4) ) = List(0 1 2 3 4)
External( func:append(List(1 1) List(1) List(1) List(List(1))) ) = L
External( func:append(List() 1) ) = List(1)
```

4.11.4.6 `func:concatenate` (adapted from `fn:concatenate`)

- *Schema*:

  `(?list1 ... ?listn; func:concatenate(?list1 ... ?listn))`

- *Domains*:

  ?list$_1$: D_{list}
  ...
  ?list$_n$: D_{list}

- *Mapping*:

Returns a list consisting of all the items in $list_1$, followed by all the items in $list_i$, for each i <= n. That is, $I_{external}$(?$list_1$ ... ?$list_n$; func:concatenate(?$list_1$ ... ?$list_n$))(l_1 ... l_n) = I_{list}($t_{1,0}$ ... t_{1,k_1} ... $t_{n,0}$ ... t_{n,k_n}) if, for each i between 1 and n, I_{list}^{-1}(l_i) = ($t_{i,0}$ ... t_{i,k_i}).

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

- *Examples*:

```
External( func:concatenate(List(0 1 2) List(3 4 5)) ) = List(0 1 2 3
External( func:concatenate(List(1 1) List(1) List(1)) ) = List(1 1 1
External( func:concatenate(List())) = List()
External( func:concatenate(List() List(1) List() List(2)) ) = List(1
```

4.11.4.7 `func:insert-before` (adapted from `fn:insert-before`)

- *Schema*:

```
(?list ?position ?newItem; func:insert-
before(?list ?position ?newItem))
```

- *Domains*:

?list: D_{list}
?position: value space of xs:int
?newItem: unrestricted

- *Mapping*:

Return a list which is ?list, except that ?newItem is inserted at the given ?position, with the item (if any) that was at that position, and all following items, shifted down one position. That is, $I_{external}$(?list ?position ?newItem; func:insert-before(?list ?position ?newItem))(l i s) = I_{list}(t_0 ... t_{i-1} s t_i ... t_n) if I_{list}^{-1}(l) = (t_0 ... t_n) and i corresponds to a position between 0 and n, as defined in Section Position Numbering.

If an argument value is outside of its domain or i does not correspond to a position between 0 and n as described in Section Position Numbering the value of the function is left unspecified.

- *Examples*:

```
External( func:insert-before(List(0 1 2 3 4) 0 99) ) = List(99 0 1 2
External( func:insert-before(List(0 1 2 3 4) 1 99) ) = List(0 99 1 2
External( func:insert-before(List(0 1 2 3 4) 5 99) ) = (unspecified)
External( func:insert-before(List(0 1 2 3 4) -1 99) ) = List(0 1 2 3
```

```
External( func:insert-before(List(0 1 2 3 4) -5 99) ) = List(99 0 1 2
External( func:insert-before(List(0 1 2 3 4) -10 99) ) = (unspecified
```

4.11.4.8 `func:remove` (adapted from `fn:remove`)

- *Schema*:

  ```
  (?list ?position; func:remove(?list ?position))
  ```

- *Domains*:

  ?list: $\boldsymbol{D}_{\text{list}}$
  ?position: value space of xs:int

- *Mapping*:

  Returns a list which is ?list except that the item at the given ?position has been removed. That is, $\boldsymbol{I}_{\text{external}}$(?list ?position; func:remove(?list ?position))(l i) = $\boldsymbol{I}_{\text{list}}(t_0 \ldots t_{i-1}\ t_{i+1} \ldots t_n)$ if $\boldsymbol{I}_{\text{list}}^{-1}(l) = (t_0 \ldots t_n)$ and i corresponds to a position between 0 and n, as defined in Section Position Numbering.

  If an argument value is outside of its domain or i does not correspond to a position between 0 and n as described in Section Position Numbering the value of the function is left unspecified.

- *Examples*:

  ```
  External( func:remove(List(0 1 2 3 4) 0) ) = List(1 2 3 4)
  External( func:remove(List(0 1 2 3 4) 1) ) = List(0 2 3 4)
  External( func:remove(List(0 1 2 3 4) 4) ) = List(0 1 2 3)
  External( func:remove(List(0 1 2 3 4) 5) ) = (unspecified)
  External( func:remove(List(0 1 2 3 4) 6) ) = (unspecified)
  External( func:remove(List(0 1 2 3 4) -1) ) = List(0 1 2 3)
  External( func:remove(List(0 1 2 3 4) -5) ) = List(1 2 3 4)
  External( func:remove(List(0 1 2 3 4) -6) ) = (unspecified)
  ```

4.11.4.9 `func:reverse` (adapted from `fn:reverse`)

- *Schema*:

```
(?list; func:reverse(?list))
```

- *Domains*:

  ?list: D_{list}

- *Mapping*:

  Return a list with all the items in ?list, but in reverse order. That is, $I_{external}$(?list; func:reverse(?list))(l) = $I_{list}(t_n \ldots t_0)$ if $I_{list}^{-1}(l) = (t_0 \ldots t_n)$.

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

- *Examples*:

```
External( func:reverse(List(0 1 2 3 4)) ) = List(4 3 2 1 0)
External( func:reverse(List(1)) ) = List(1)
External( func:reverse(List()) ) = List()
```

4.11.4.10 `func:index-of` (adapted from `fn:index-of`)

- *Schema*:

```
(?list ?matchValue; func:index-of(?list ?matchValue))
```

- *Domains*:

  ?list: D_{list}
  ?matchValue: unrestricted

- *Mapping*:

  Returns the ascending list of all integers, i>=0, such that External(func:get(?list,i)) = ?matchValue. That is, $I_{external}$(?list ?matchValue; func:index-of(?list ?matchValue))(l v) = $I_{list}(i_1 \ldots i_k)$ if $I_{list}^{-1}(l) = (t_0 \ldots t_n)$ such that $(i_1 \ldots i_k)$ is the ordered list of positions (as defined in Section Position Numbering) between 0 and n with $t_{i_1} = \ldots = t_{i_k} = v$.

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

- *Examples*:

```
External( func:index-of(List(0 1 2 3 4) 2) ) = List(2)
External( func:index-of(List(0 1 2 3 4 5 2 2) 2) ) = List(2 6 7)
External( func:index-of(List(2 2 3 4 5 2 2) 2) ) = List(0 1 5 6)
External( func:index-of(List(2 2 3 4 5 2 2) 1) ) = List()
```

4.11.4.11 `func:union` (inspired by `fn:union`)

- *Schema*:

  (?list$_1$ ... ?list$_n$; func:union(?list$_1$ ... ?list$_n$))

- *Domains*:

  ?list-1: ***D***list
  ...
  ?list-n: ***D***list

- *Informal Mapping*:

  Returns a list containing all the items in ?list$_1$, ..., ?list$_n$, in the same order, but with all duplicates removed.

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

- *Note*:

  External(func:union(list$_1$ ... list$_n$)) is equivalent to
  External(func:distinct_values(External(
  func:concatenate(list$_1$ ... list$_n$)))).

- *Examples*:

```
External( func:union(List(0 1 2 4) List(3 4 5 6)) ) = List(0 1 2 4 3
External( func:union(List(0 1 2 3) List(4)) ) = List(0 1 2 3 4)
External( func:union(List(0 1 2 3) List(3)) ) = List(0 1 2 3)
```

  External(func:union(List(0 2 1 0))) = List(0 2 1)

4.11.4.12 `func:distinct-values` (adapted from `fn:distinct-values`)

- *Schema*:

```
(?list; func:distinct-values(?list))
```

- *Domains*:

?list: D_{list}

- *Informal Mapping*:

  Returns a list which contains exactly those items which are in ?list, in the order of first appearance, except that all except the first occurance of any item are deleted.

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

- *Examples*:

```
External( func:distinct-values(List(0 1 2 3 4)) ) = List(0 1 2 3 4)
External( func:distinct-values(List(0 1 2 3 4 0 4)) ) = List(0 1 2 3
External( func:distinct-values(List(3 3 3)) ) = List(3)
```

4.11.4.13 `func:intersect` (inspired by `fn:intersect`)

- *Schema*:

  $(?list_1 \ldots ?list_n;$ func:intersect$(?list_1 \ldots ?list_n))$

- *Domains*:

  ?$list_1$: D_{list}
  ...
  ?$list_n$: D_{list}

- *Informal Mapping*:

  Returns a list which contains exactly those items which are common to all argument lists. The order of the items in the returned list is the same as the order in ?$list_1$.

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

- *Examples*:

```
External( func:intersect(List(0 1 2 3 4) List(1 3)) ) = List(1 3)
External( func:intersect(List(0 1 2 3 4) List(3 1)) ) = List(1 3)
External( func:intersect(List(0 1 2 3 4) List()) ) = List()
External( func:intersect(List(0 1 2 3 4) List(0 1 2 3 4 5 6)) ) = Lis
```

4.11.4.14 `func:except` (inspired by `fn:except`)

- *Schema*:

  `(?list1 ?list2; func:except(?list1 ?list2))`

- *Domains*:

  ?list1: ***D***list
  ?list2: ***D***list

- *Informal Mapping*:

  Returns a list which contains exactly those items which are in ?list1 and not in ?list2. The order of the items is the same as in ?list1.

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

- *Examples*:

```
External( func:except(List(0 1 2 3 4) List(1 3)) ) = List(0 2 4)
External( func:except(List(0 1 2 3 4) List()) ) = List(0 1 2 3 4)
External( func:except(List(0 1 2 3 4) List(0 1 2 3 4)) ) = List()
```

5 References

[BCP-47]
BCP 47 - Tags for the Identification of Languages, A. Phillips, M. Davis, IETF, Sep 2006, http://tools.ietf.org/html/bcp47.

[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-PLAINLITERAL]
`rdf:PlainLiteral`: A Datatype for RDF Plain Literals, J. Bao, S. Hawke, B. Motik, P.F. Patel-Schneider, A. Polleres (Editors), W3C Candidate Recommendation, 11 June 2009, http://www.w3.org/TR/2009/CR-rdf-plain-literal-20090611/. Latest version available at http://www.w3.org/TR/rdf-plain-literal/.

[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.

[RIF-BLD]
RIF Basic Logic Dialect Harold Boley, Michael Kifer, eds. W3C Candidate Recommendation, 1 October 2009, http://www.w3.org/TR/2009/CR-rif-bld-20091001/. Latest version available at http://www.w3.org/TR/rif-bld/.

[RIF-Core]
RIF Core Dialect Harold Boley, Gary Hallmark, Michael Kifer, Adrian Paschke, Axel Polleres, Dave Reynolds, eds. W3C Candidate Recommendation, 1 October 2009, http://www.w3.org/TR/2009/CR-rif-core-20091001/. Latest version available at http://www.w3.org/TR/rif-core/.

[RIF-FLD]
RIF Framework for Logic Dialects Harold Boley, Michael Kifer, eds. W3C Candidate Recommendation, 1 October 2009, http://www.w3.org/TR/2009/CR-rif-fld-20091001/. Latest version available at http://www.w3.org/TR/rif-fld/.

[RIF-PRD]
RIF Production Rule Dialect Christian de Sainte Marie, Adrian Paschke, Gary Hallmark, eds. W3C Candidate Recommendation, 1 October 2009, http://www.w3.org/TR/2009/CR-rif-prd-20091001/. Latest version available at http://www.w3.org/TR/rif-prd/.

[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/.

[SWRL]
SWRL: A Semantic Web Rule Language Combining OWL and RuleML, I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B., M. Dean, W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/. Latest version available at http://www.w3.org/Submission/SWRL/.

[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 Schema Datatypes]
W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes. David Peterson, Shudi Gao, Ashok Malhotra, C. M. Sperberg-McQueen, and Henry S. Thompson, eds. W3C Candidate Recommendation, 30 April 2009, http://www.w3.org/TR/2009/CR-xmlschema11-2-20090430/. Latest version available as http://www.w3.org/TR/xmlschema11-2/.

[XPath]
XML Path Language (XPath) 2.0, A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M, Kay, J. Robie, J. Siméon (Editors), W3C Recommendation, World Wide Web Consortium, 23 January 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/. Latest version available at http://www.w3.org/TR/xpath20/.

[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/.

6 Appendix: Schemas for Externally Defined Terms

The RIF Framework for Logic Dialects introduces a general notion of externally defined terms and their schemes. However, RIF-BLD and the present document use only restricted kinds of external terms. To make this document self-contained, this appendix provides a complete description of these restricted notions.

In RIF-FLD, an external term is an expression of the form `External(id τ)`, where `id` is a term that identifies the source that defines the term `τ` and `τ` itself can be a constant, a positional or named-arguments term, a frame, an equality, or a classification term. In RIF-BLD, only positional and named-argument terms are allowed as `τ`, and RIF-DTB builtins can only be positional terms. So, only a ***restricted kind of external terms*** is used: `External(τ)`, where `τ` has one of the aforementioned forms. If `τ` is a term of the form `p(...)` then `External(τ)` is treated as a shorthand for `External(p τ)`, but this extended 2-argument form of `External` itself is not allowed in RIF-BLD.

External schemas 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.

Definition (Schema for external term). An ***external schema*** is a statement of the form $(?X_1 \ldots ?X_n; \tau)$ where

- τ is a positional or a named-argument term.
- $?X_1 \ldots ?X_n$ 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.

Note that RIF-FLD defines external schemas as triples $(id; ?X_1 \ldots ?X_n; \tau)$, where `id` is the identifying term for the schema's source. However, since RIF-BLD uses a simplified version of externally defined terms in which `id` is determined by the predicate/function name in `τ`, the `id`-part is omitted in the above simplified version of external schemas.

A term `t` is an ***instantiation*** of an external schema $(?X_1 \ldots ?X_n; \tau)$ iff `t` can be obtained from `τ` by a simultaneous substitution $?X_1/s_1 \ldots ?X_n/s_n$ of the variables $?X_1 \ldots ?X_n$ with terms $s_1 \ldots s_n$, respectively. Some of the terms s_i can be variables themselves. For example, `?Z[foo->f(a ?P)]` is an instantiation of `(?X ?Y; ?X[foo->?Y])` by the substitution `?X/?Z ?Y/f(a ?P).` □

Observe that a variable cannot be an instantiation 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 instantiations 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 instantiation 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.

7 Appendix: Changes from Last Call Version of July 3, 2009

- Change log section added
- A number of typos were found and fixed.
  - String-join mapping definition was fixed to correct definition following XPath (original subscripts were incorrect).
  - Removed stray ? in mapping definition of substring-before
  - Added heading for Mapping section of pred:duration-equal
- "instance" of an external schema was replaced with "instantiation" of an external schema in Appendix.
- Position numbering was made At Risk
- More formal definitions for some list predicates and functions were added
- Type promotion for numeric types was made explicit
- An explanitory note was added on negative guards
- Type hierarchy was fixed
- concat was made to require two or more arguments, as in xpath
- The start-before-stop case of sublist was explicitely declared undefined
- union, intersection, and except were called "inspired by" instead of "adapted from" xpath, because they are so different.