TeXQuery: A Full-Text Search Extension to XQuery

Part II – Formal Semantics

Sihem Amer-Yahia
sihem@research.att.com

Chavdar Botev
cbotev@cs.cornell.edu

Jonathan Robie
jonathan.robie@datadirect-technologies.com

Jayavel Shanmugasundaram
jai@cs.cornell.edu

1 AT&T Labs
2 Computer Science Department, Cornell University
3 DataDirect Technologies

14 July 2003
## Contents

1. **Introduction** ................................................................. 3

2. **Nested XQuery Expressions in FTSelections** ................................. 4
   2.1. FTStringSelection .................................................. 4
   2.2. FTRangeSpec ......................................................... 4
   2.3. FTStopWordsCtxMod ............................................. 4
   2.4. FTThesaurusCtxMod .............................................. 4
   2.5. TFLanguageCtxMod ............................................... 4
   2.6. FTIgnoreCtxMod .................................................. 4

3. **The FullMatch Data Model** ............................................. 5
   3.1. Positions ............................................................... 5
   3.2. FullMatch ............................................................... 6
       3.2.1. Examples and Intuition ......................................... 6
       3.2.2. Formal Model .................................................. 8
       3.2.3. XML Representation of a FullMatch ...................... 8

4. **Semantics of TeXQuery Expressions** .................................... 11
   4.1. Implementation-defined functions ................................ 11
   4.2. Semantics of FTContainsExpr ................................... 11
   4.3. Semantics of FTScoreExpr ........................................ 12
   4.4. Semantics of FTSearchExpr ...................................... 13

5. **Semantics of FTSelections** ............................................ 15
   5.1. “Evaluate” Function: High-Level Description .................. 15
   5.2. Semantics of ApplyFTSelection Function for each FTSelection ... 18
       5.2.1. Implementation-defined functions ......................... 18
       5.2.2. Semantics of FTStringSelection .......................... 19
       5.2.3. FTOrConnective ............................................... 20
       5.2.4. FTAndConnective ............................................. 22
       5.2.5. FTNegation ..................................................... 23
       5.2.6. FTMildNegation ............................................... 25
       5.2.7. FTOrderSelection ............................................ 27
       5.2.8. FTScopeSelection ........................................... 29
       5.2.9. FTDistanceSelection ....................................... 32
       5.2.10. FTWindowSelection ........................................ 40
       5.2.11. FTTimesSelection .......................................... 44

6. **Example** ........................................................................ 47

7. **References** ................................................................... 64
1. Introduction

This document describes the formal semantics of the TeXQuery extension to XQuery. As discussed in Part I – Language Specification and shown in Figure 1, **FTSelections** are the basic expression in a full-text query. **FTSelections** are fully composable in the sense that each **FTSelection** operates on zero or more **FullMatches** and returns a **FullMatch**. Regular XQuery expressions can be nested inside **FTSelections**, and **FTSelections** can themselves be nested inside regular XQuery expressions.

![Figure 1 Composing XQuery and TeXQuery expressions](image)

Given the above relationship between XQuery expressions and **FTSelections**, this formal semantics document addresses the following issues. First, it defines the semantics of XQuery expression nested in **FTSelections** (bottom arrow in Figure 1). Second, it defines the notion of a **FullMatch** and specifies how each **FTSelection** operates on zero or more **FullMatches** and returns a **FullMatch** (right arrow in Figure 1). Finally, it specifies how TeXQuery expressions convert a **FullMatch** to a sequence of nodes/atoms in the XQuery data model (top arrow in Figure 1). The semantics of XQuery expressions (left arrow in Figure 1) is specified in the XQuery semantics document [1] and is not discussed here.

The rest of this document is organized as follows. In Section 2, we specify the semantics of nesting XQuery expressions in **FTSelections**. In Section 3, we describe the **FullMatch** data model. In Section 4, we describe how TeXQuery expressions convert a **FullMatch** to a sequence of nodes/atoms. In Section 5, we specify the semantics of **FTSelection** expressions. First (5.1), the general FTSelection expression semantics is described. Then in 5.2, we present the semantics of individual FTSelection expressions. Finally, in Section 6, we present a complete example that illustrates the main points from the previous sections.
2. Nested XQuery Expressions in FTSelections

The semantics of XQuery expressions nested inside FTSelections is given below.

2.1. FTStringSelection

The XQuery expression nested inside an FTStringSelection must evaluate to a sequence of string values after applying atomization [1] (otherwise the entire FTSelection returns an error). The sequence of strings is used as described in Part I.

2.2. FTRangeSpec

The XQuery expression (or expressions, in the case of a “from-to” range) must evaluate to a singleton sequence of integers after applying atomization [1] (otherwise the entire FTSelection returns an error). The resulting integer values are treated as boundaries for the corresponding range as described in Part I.

2.3. FTStopWordsCtxMod

The XQuery expression must evaluate to a sequence of string values after applying atomization [1] (otherwise the entire FTSelection returns an error). The resulting string values are treated as stop words that must be ignored during phrase matching and proximity evaluation as discussed in Part I.

2.4. FTThesaurusCtxMod

The XQuery expression sub-expression must evaluate to a sequence of string values after applying atomization [1] (otherwise the entire FTSelection returns an error). The resulting string values are treated as names of thesauri to use during string matching as discussed in Part I.

2.5. FTLanguageCtxMod

The XQuery sub-expression must evaluate to either an empty sequence or a singleton sequence of a string value or an empty sequence after applying atomization [1] (otherwise the entire FTSelection returns an error). The resulting string value is treated as a language identifier specifying the language of the matched document/documents as discussed in Part I.

2.6. FTIgnoreCtxMod

The XQuery sub-expression must evaluate to a sequence of element nodes (otherwise the entire FTSelection returns an error). The resulting element nodes define the nodes whose element tags/content must be ignored as discussed in Part I.
3. The FullMatch Data Model

As described in Part I, the XQuery data model of a “sequence of nodes” is inadequate for fully composable \textit{FTSelections}. The main reason is that full-text operations (such as \textit{FTSelections}) operate on linguistic tokens, such as positions of words, and such information is not captured in the XQuery data model. We thus define the \textit{FullMatch} data model that allows for fully compositional \textit{FTSelections}. Before formally defining the \textit{FullMatch} data model, we first describe the key concept of a position.

3.1. Positions

\textit{A position is the identity of a linguistic token inside an XML document.} Each position is associated with:

\begin{itemize}
  \item the linguistic token it identifies,
  \item the relative position of the linguistic token in the document,
  \item the relative position of the sentence containing the linguistic token
  \item the relative position of the paragraph containing the linguistic token
  \item the node that directly contains the linguistic token, and
  \item where the linguistic token appears in the node containing it (i.e., tag name, element content, etc.)
\end{itemize}

As an illustration, consider the following XML fragment:

\begin{verbatim}
<offer id="1000" price="10000">
  Ford Mustang 2000, 65K, excellent condition, runs great, AC, CC, power all
</offer>
<offer id="1001" price="8000">
  Honda Accord 1999, 78K, A/C, cruise control, runs and looks great, excellent condition
</offer>
<offer id="1005" price="5500">
  Ford Mustang, 1995, 150K highway mileage, no rust, excellent condition
</offer>
\end{verbatim}

If we assume that linguistic tokens are delimited by punctuation and whitespace symbols (in English), the first linguistic token “offer” (the element tag name) will be assigned a relative position of 1, the linguistic token “id” (the attribute name) will be assigned a relative position of 2, the linguistic token “100” (the value of attribute id) will be assigned a relative position of 2, and so on. The relative positions of the linguistic tokens are shown below in parenthesis.

\begin{verbatim}
<offer(1) id(2)="1000(3)" price(4)="10000(5)">
  Ford(6) Mustang(7) 2000(8), 65K(9), excellent(10) condition(11), runs(12) great(13), AC(14), CC(15),
  power(16) all(17)
</offer(18)>
<offer(19) id(20)="1001(21)" price(22)="8000(23)">
\end{verbatim}
Honda (24) Accord (25) 1999 (26), 78K (27), A (28) / C (29), cruise (30) control (31), runs (32) and (33) looks (34) great (35), excellent (36) condition (37) </offer (38)>

<offer (39) id (40) = "1005 (41) " price (42) = "5500 (43) ">
Ford (44) Mustang (45), 1995 (46), 150K (47) highway (48) mileage (50), little (60) rust (61), excellent (62) condition (63) </offer (64)>

The relative positions of paragraphs are determined similarly. Assuming that the paragraph delimiters are start tag (“<”), end tag (“>”), and end of line characters, the first tag will be assigned a paragraph relative number 1, the following element content will be assigned a relative number 2, the end tag will be assigned relative number 3, and so on.

The relative positions of sentences are also determined similarly using sentence delimiters such as “.”, “!” and “?”.

3.2. FullMatch

We now define a FullMatch. Intuitively, a FullMatch specifies the positions that a node should contain, and the positions that a node should not contain, in order to satisfy an FTSelection. We first introduce a FullMatch using some examples and intuition. We then formally define a FullMatch.

3.2.1. Examples and Intuition

Consider the FTStringSelection “Mustang” evaluated over the sample document fragment in the previous section. The FullMatch corresponding to this FTStringSelection is shown in Figure 2. As shown, the FullMatch consists of two SimpleMatches. Each SimpleMatch represents one possible “solution” to the FTStringSelection “Mustang”. The “solution” to the first SimpleMatch are those nodes that contain (represented as StringInclude) the linguistic token “Mustang” at position 7. The “solution” to the second SimpleMatch are those nodes that contain the linguistic token “Mustang” at position 45.

Note that a FullMatch does not directly list the nodes that satisfy an FTSelection – rather, it specifies a position-based “predicate” that nodes need to satisfy in order to qualify as a solution to an FTSelection. By specifying a FullMatch in terms of positions (a linguistic token notion) rather than nodes, there is

Figure 2 Sample Full Match
sufficient information in a FullMatch to achieve full compositionality among FTSelections. At the same time, the interpretation of a FullMatch as a predicate on nodes preserves the mapping to the XQuery data model, and allows a FullMatch to be mapped back to a sequence of nodes when necessary.

Let us now consider a more complex example. Consider the FTStringSelection “Ford Mustang” evaluated over the XML fragment used above. The FullMatch for this FTStringSelection is shown on Figure 3. There are two possible “solutions” to this FTStringSelection, and these are represented by the two SimpleMatches. Each of the SimpleMatches requires two linguistic tokens to be matched. The “solution” corresponding to the first SimpleMatch is obtained by matching “Ford” at position 6 and matching “Mustang” at position 7. Similarly, the “solution” to the second SimpleMatch is obtained by matching “Ford” at position 44 and “Mustang” at position 45.

The observant reader may have noticed that the FullMatch structure resembles the Disjunctive Normal Form (DNF) in propositional and first-order logic. A node is a “solution” to a FullMatch iff it is a solution to at least one of its SimpleMatches – this is similar to a DNF formula being satisfied iff at least one of its disjuncts is satisfied. A node is a “solution” to a SimpleMatch iff all of its StringIncludes are satisfied – this is similar to a disjunct in a DNF formula being true iff all of its atomic terms evaluate to true. The analogy of a FullMatch to a DNF formula is a very useful one, and we will use this analogy to help illustrate many of the compositionality properties of FullMatch in later sections.

Let us now consider a more sophisticated example of a FullMatch. Consider the FTSelection “Mustang” \&\& ! “rust” that searches for nodes that contain “Mustang” but not “rust”. The FullMatch for this FTSelection is shown in Figure 4. Observe the new type of component: StringExclude. This is the component that corresponds to negation – it specifies that the “solution” to the corresponding simple match should not match the linguistic token at the specified position. For instance, the first SimpleMatch specifies the “solution” that “Mustang” should be matched at position 7, and “rust” should not be matched at position 61.
Note that the idea of StringExclude also has a direct analogy in a DNF formula – a StringExclude corresponds to the negation of an atom in a disjunct.

3.2.2. Formal Model

We are now ready to present the formal model of a FullMatch. The UML Static Class diagram of a FullMatch is shown in Figure 5. A FullMatch contains zero or more SimpleMatches. A SimpleMatch contains zero or more StringIncludes and zero or more StringExcludes. Both StringInclude and StringExclude are of type StringMatch. The queryString attribute of specifies the linguistic token associated with StringMatch. The queryPos attribute specifies the position of the corresponding search token in the query (the need for this field will be apparent in later sections). The position attribute is the position where the linguistic token was matched.

Intuitively, a FullMatch encodes all possible “solutions” to a FTSelection. Each individual solution is encoded in a SimpleMatch. The SimpleMatch can specify that certain linguistic tokens at specified positions should be included (the stringInclude component) and/or specify that certain linguistic tokens at specified positions should not be included (stringExclude).

3.2.3. XML Representation of a FullMatch

FullMatch has a well-defined hierarchical structure as shown in Figure 5. Therefore, a FullMatch can be easily modeled in XML. In subsequent sections, we will use this XML representation to formally describe the semantics of FTSelections. In particular, we will use the XML representation of a FullMatch to formally specify how an FTSelection operates on zero or more FullMatches to produce a resulting
FullMatch. We will also use the XML representation to specify the formal semantics of the TeXQuery expressions.

A simple XML schema for modeling FullMatches is given below:

```xml
<xs:complexType name="fts:FullMatch">
  <xs:sequence>
    <xs:element name="simpleMatch" type="fts:SimpleMatch" minOccurrences="1" maxOccurrences="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="fts:SimpleMatch">
  <xs:sequence>
    <xs:element name="stringInclude" type="fts:StringMatch" minOccurrences="0" maxOccurrences="unbounded" />
    <xs:element name="stringExclude" type="fts:StringMatch" minOccurrences="0" maxOccurrences="unbounded" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="fts:StringMatch">
  <xs:attribute name="queryPos" type="xs:integer" />
  <xs:attribute name="queryString" type="xs:string" />
  <xs:sequence>
    <xs:element name="docPos" type="fts:Position" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="fts:Position">
  <xs:attribute name="abs" type="xs:integer" />
  <xs:attribute name="para" type="xs:integer" />
  <xs:attribute name="sentence" type="xs:integer" />
  <xs:attribute name="term" type="xs:string" />
  <xs:element name="elem" type="Node" />
</xs:complexType>

The FullMatch in Figure 4 can be represented as the following XML document conforming the XML schema:

```xml
<fts:fullMatch>
  <fts:simpleMatch>
    <fts:stringInclude />
    <fts:stringExclude />
  </fts:simpleMatch>
</fts:fullMatch>
```
<fts:stringExclude queryString="rust" queryPos="2">
  <pos @abs="61" @para="8" @sentence="8" term="rust">
    <elem id="..." />
  </pos>
</fts:stringExclude>
<fts:stringInclude queryString="Mustang" queryPos="1">
  <pos @abs="7" @para="2" @sentence="2" term="Mustang">
    <elem id="..." />
  </pos>
</fts:stringInclude>
</fts:simpleMatch>
<fts:simpleMatch>
  <fts:stringExclude queryString="rust" queryPos="2">
  </fts:stringExclude>
</fts:simpleMatch>
<fts:simpleMatch>
  <fts:stringInclude queryString="Mustang" queryPos="1">
    <pos @abs="45" @para="8" @sentence="8" term="Mustang">
      <elem id="..." />
    </pos>
  </fts:stringInclude>
</fts:simpleMatch>
</fts:fullMatch>
4. Semantics of TeXQuery Expressions

We now present the formal semantics of TeXQuery expressions. Recall from Part I that there are three TeXQuery expressions: FTContainsExpr, ScoreByExpr, and FTSearchExpr. Each of these expressions takes in (1) an evaluation context consisting of a sequence of nodes (which is the result of a regular XQuery expression), and (2) a FullMatch corresponding to an FTSelection, and returns a sequence of nodes. Since TeXQuery expressions return results in the XQuery data model (a sequence of nodes), TeXQuery expressions can be treated like regular XQuery expressions and can be fully composed with other XQuery expressions. In addition, since TeXQuery expressions also map from FullMatches to a sequence of nodes, they provide the “glue” and well-defined semantics for mapping from the FTSelection data model and the XQuery data model.

The formal semantics of TeXQuery expressions is specified in terms of two functions. How these two functions are computed is implementation-defined, but the functions have to satisfy some well-defined properties. We first present the properties of the implementation-defined functions, and then present the semantics of TeXQuery expressions based on these functions.

4.1. Implementation-defined functions

The following two functions must be defined by any compliant TeXQuery implementation:

$$\text{function } \text{fts:containsPos}(\$node \text{ as Node, } \$pos \text{ as fts:Position}) \text{ as xs:boolean}$$

fts:containsPos returns true iff the position $\$pos$ is contained (directly or indirectly) in node $\$node$

$$\text{function } \text{fts:score}(\$node \text{ as Node, } \$ftselection \text{ as FTSelectionWithScoreWeights}) \text{ as xs:double}$$

fts:score returns the score of node $\$node$ given an FTSelectionWithScoreWeights. The exact nature of how the score is computed is implementation-defined and can use score weights in the FTSelectionWithScoreWeights. The function, however, should satisfy the following two properties:

1) The score returned should be a floating point number in the range (0, 1], and
2) A higher score should indicate a higher degree of relevance to the FTSelectionWithScoreWeights

4.2. Semantics of FTContainsExpr

Recall from Part I that an FTContainsExpr is of the form “EvaluationContext ftcontains FTSelection1”, where EvaluationContext is an XQuery expression that returns a sequence of nodes, and FTSelection1 is an FTSelection that returns a FullMatch. Intuitively, the FTContainsExpr returns true iff some node in the result of EvaluationContext satisfies the FullMatch returned by FTSelection1.

We now formally define the semantics of FTContainsExpr. The semantics is defined in terms of a regular XQuery function (without any TeXQuery extensions). The XQuery function takes in two parameters: the first parameter is the sequence of nodes returned by EvaluationContext, and the second parameter is the XML node representation of the FullMatch returned by FTSelection1 (see Section 3.2.3). The XQuery function (by definition) returns true iff the corresponding FTContainsExpr returns true, and thus specifies the semantics of FTContainsExpr. Note that by using regular XQuery to specify the formal semantics, we avoid the need to introduce new formalism – we simply reuse the formal semantics of XQuery.

$$\text{define function}$$

$$\text{FTContainsExpr}(\$evaluationContext \text{ as Node*}, \$$
Intuitively, the above function returns true iff some node in the evaluation context satisfies at least one of the SimpleMatches. The function that defines when a node satisfies a SimpleMatch (satisfiesSimpleMatch) is defined below.

```
define function
satisfiesSimpleMatch($node as Node,
    $simpleMatch as element(simpleMatch,
        fts:SimpleMatch)
) as xs:Boolean {

    return (every $stringInclude in $simpleMatch/stringInclude
        satisfies fts:containsPos($node, $stringInclude/pos)
    )
    and
    (every $stringExclude in $simpleMatch/stringExclude
        satisfies not fts:containsPos($node, $stringInclude/pos)
    )
}
```

Intuitively, the above function returns true iff the node contains all the StringInclude positions, and does not contain all the StringExclude positions. Note that fts:containsPos is an implementation-defined function whose semantics is defined in Section 4.1.

### 4.3. Semantics of FTScoreExpr

Recall from Part I that a FTScoreExpr is of the form “EvaluationContext ftscore FTSelectionWithScoreWeights1”, where EvaluationContext is an XQuery expression that returns a sequence of nodes, and FTSelectionWithScoreWeights1 is an FTSelectionWithScoreWeights that returns a FullMatch and also specifies score weights. Intuitively, FTScoreExpr returns a sequence of scores corresponding to each node in the evaluation context, where each score is computed using the specification in FTSelectionWithScoreWeights1. The XQuery function defining this semantics is given below.

```
define function
FTScoreExpr($evaluationContext as Node*,
    $fullMatch as element(fullMatch,
        fts:FullMatch)
)
```
For every node in the evaluation context, if the node satisfies the `FullMatch` (checked using `FTContainsExpr` function defined earlier), then a positive score computed using the implementation-defined function `fts:score` is returned. Else a zero score is returned.

### 4.4. Semantics of `FTSearchExpr`

Recall from Part I that an `FTSearchExpr` is of the form “`EvaluationContext ftsearch FTSelectionWithScoreWeights1`”, where `EvaluationContext` is an XQuery expression that returns a sequence of nodes, and `FTSelection1` is an `FTSelection` that returns a `FullMatch`. Intuitively, the `FTSearchExpr` returns the most-specific nodes in the result of `EvaluationContext` or its descendants that satisfy the `FullMatch` returned by `FTSelection1`.

The formal semantics of `FTSearchExpr` is specified by the following XQuery function.

```xquery
define function FTSearchExpr($evaluationContext as Node*,
    $fullMatch as element(fullMatch,
        fts:FullMatch),
    $ftSelection as FTSelectionWithScoreWeights
) as Node* {

    for $node in $evaluationContext/descendant-or-self::node()
        where FTContainsExpr($node, $fullMatch)
            and every $descendant in $node/descendant::node()
                satisfies not FTContainsExpr($descendant, $fullMatch)
        order by fts:score($node, $ftSelection) descending
    return $node
}
```

The above function first determines all the descendants nodes of the evaluation context (including the evaluation context nodes) in `$node`. It then returns only those `$node` nodes that (a) satisfy the `FullMatch`, and (b) do not have any descendants that satisfy the `FullMatch` (checking for `FullMatch` satisfaction is done by a call to the `FTContainsExpr` function defined earlier). By doing so, the function ensures that only the most specific results for a `FullMatch` are returned – in particular, if a node is returned by
ftsearch, none of its ancestors will be returned. The result nodes are returned sorted in descending order of their score with regards to the $FTSelectionWithScoreWeights$ expression.
5. Semantics of FTSelections

In this section, we define the semantics of FTSelections. Recall from Part I and Figure 1 (right arrow) that FTSelections are fully composable, and can be arbitrarily nested under other FTSelections. Also, each FTSelection can be associated with context modifiers (such as stemming, stop words, etc.) and score weights. Since score weights are solely interpreted by the implementation-defined scoring function (described in Section 4.1), score weights do not influence the semantics of FTSelections in any way. We will thus not consider score weights when defining the formal semantics.

We now present operational semantics for the evaluation of FTSelections. Specifically, we define a function “evaluate” that takes in three parameters: (1) an FTSelection, (2) an evaluation context (specified by the TeXQuery expression the FTSelection is nested under), and (3) the default (implementation-defined) set of context modifiers that apply to the evaluation of the FTSelection. The “evaluate” function returns the FullMatch that is the result of evaluating the FTSelection. The “evaluate” function works by recursively calling itself on nested FTSelections, which in turn return FullMatches. Then, the “evaluate” function calls the polymorphic function “applyFTSelection” that implements the evaluation of the particular FTSelection applied on the FullMatches returned by the evaluation of the nested FTSelections. Thus full compositionality of FTSelections is achieved as depicted in the right arrow in Figure 1.

We first present a high-level description of the “evaluate” function, and then describe the details.

5.1. “Evaluate” Function: High-Level Description

The high-level pseudo-code (not XQuery function) for the “evaluate” function is given below.

```plaintext
function evaluate(ftSelection: FTSelection,
  evaluationContext: EvaluationContext,
  modifierContext: Stack)
returns FullMatch
begin
  switch (ftSelection)
  case (nftSelection: FTSelection   modifier: FTContextModifier):
    // This is a new context modifier for the
    // nested FTSelections (nftSelection). So,
    // create new modifier object and add it to the
    // top of the current modifier context
    modifierItem := createModifierItem(modifier);
    modifierContext.push(modifierItem);
    resultFullMatch := evaluate(nftSelection,
      evaluationContext,
      modifierContext);
    modifierContext.pop();
    return resultFullMatch;
```
case (nftSelection: FTSelection “weight” w: FTWeight):
    // Weight has no bearing on semantics – just
    // call “evaluate” on nested FTSelection
    return evaluate(nftSelection,
                    evaluationContext,
                    modifierContext);

case (ftSelection: FTStringSelection):
    // Apply the FTStringSelection in the evaluation
    // context
    return applyFTSelection(evaluationContext,
                             modifierContext,
                             ftSelection.SearchToken,
                             ftSelection.queryPos);

    // All FTSelections except FTContextModifier, FTWeight and
    // FTStringSelection
    case (ftSelection: FTAndConnective) |
          (ftSelection: FTOrConnective) |
          (ftSelection: FTNegation) |
          (ftSelection: FTMildNegation) |
          (ftSelection: FTOrderSelection) |
          (ftSelection: FTScopeSelection) |
          (ftSelection: FTDistanceSelection) |
          (ftSelection: FTWindowSelection) |
          (ftSelection: FTTimesSelection):
          // First evaluate nested FTSelections
          for (each nested FTSelection nftSelectioni) do
              fullMatchi := evaluate(nftSelectioni,
                                     evaluationContext,
                                     modifierContext);
          endfor

          // Now transform nested FullMatches into
          // result FullMatch. Note that different
          // types of FTSelections have different
          // transformations
Let us now walk through the above pseudo-code to understand the semantics of the function. For concreteness, let us assume that the *FTSelection* was invoked inside an ftcontains expression such as “*EvaluationContext ftcontains FTSelection1*” (of course, the same description carries over to ftsearch and ftscore as well). In order to determine the *FullMatch* result of *FTSelection1*, the “evaluate” function is invoked as follows: evaluate(*FTSelection1, EvaluationContext, ModifierContext*).

The *ModifierContext* above is the default (implementation-defined) list of modifiers that apply to the evaluation of *FTSelection1* (such as stemming but not thesaurus) and is implementation-defined. Context modifiers embedded in *FTSelection1* can change the modifier context as evaluation proceeds. In order to express the order in which modifiers are applied to an *FTSelection*, the modifiers are organized in a stack. The top modifier in the stack is to be applied first, the next modifier is to be applied second, and so on. The ordering among modifiers is necessary because modifiers are not always commutative — for example, synonym(stem(linguistic token)) is not always the same as stem(synonym(linguistic token)). Of course, modifiers can be reordered when they commute, but this is an optimization issue and is beyond the scope of this semantics document.

Given the invocation of: evaluate(*FTSelection1, EvaluationContext, ModifierContext*), evaluation proceeds as follows. First, *FTSelection1* is checked to see whether it is a context modifier applied on a nested *FTSelection* (case 1), a weight specification (case 2), a *FTStringSelection* (case 3), or some other *FTSelection* (case 4). Let us consider these three cases in turn.

**Case 1**: If *FTSelection1* contains a context modifier, then it modifies the context for the nested *FTSelection*. Consequently, a new context modifier element is created and pushed onto the top of the stack of context modifiers. The createModifierElement function used to create a stack element corresponding to the modifier simply creates a data structure that stores the type of modifier (such as stemming, thesaurus, synonyms, ignore, etc.) and the details relating to the modifier (such as the name of the thesaurus, the words to ignore, etc.). The context modifier created is added to the top of the stack because, in the *FTSelection*, it was applied before the other modifiers in the current modifier context. The “evaluate” function is then invoked on the nested *FTSelection* with the new modifier context. When the function returns, the modifier is popped from the stack, and the result of the nested “evaluate” function is returned. The modifier is popped because the modifier context should not apply to *FTSelections* outside its scope.

**Case 2**: If *FTSelection1* contains a weight specification, then the specification is simply ignored (because it does not alter semantics). The “evaluate” function is recursively called on the nested *FTSelection* and the resulting *FullMatch* is directly returned.

**Case 3**: If *FTSelection1* is a *FTStringSelection*, then it does not have any nested *FTSelections*. Consequently, this is the base of the recursive call, and the *FullMatch* result of the *FTStringSelection* is computed and returned. The *FullMatch* is computed by invoking the applyFTSelection function with the current evaluation context and other necessary information. The semantics of how exactly applyFTSelection creates a *FullMatch* for *FTStringSelection* will be specified in the next section.

**Case 4**: If *FTSelection1* contains neither a context modifier nor a weight specification and is not a *FTStringSelection*, the *FTSelection* performs some form of full-text operation such as ‘&&’, ‘||’,
‘window’, etc. Note that these operations are fully-compositional, and can be invoked on nested \textit{FTSelections}. Consequently, evaluation proceeds as follows. First, the “evaluate” function is recursively invoked on each nested \textit{FTSelection}. The result of evaluating each nested \textit{FTSelection} is a \textit{FullMatch}. These \textit{FullMatches} are transformed into a result \textit{FullMatch} by applying the full-text operation corresponding to \textit{FTSelection1} (generically called \textit{applyFTSelection} in the pseudo-code). As an example, let \textit{FTSelection1} be \textit{FTSelection2} && \textit{FTSelection3}. Here \textit{FTSelection2} and \textit{FTSelection3} can themselves be arbitrarily nested \textit{FTSelections}. Thus, evaluate is invoked on \textit{FTSelection2} and \textit{FTSelection3}, and the resulting \textit{FullMatches} are transformed to the output \textit{FullMatch} using the \textit{applyFTSelection} function corresponding to ‘&&’.

Note that specifying the semantics of the \textit{applyFTSelection} function for each \textit{FTSelection} is key to specifying the semantics of the \textit{FTSelection} itself. In the subsequent sections, we define the semantics of the \textit{applyFTSelection} function for each \textit{FTSelection}.

\section{Semantics of \textit{ApplyFTSelection} Function for each \textit{FTSelection}}

We now define the semantics of the \textit{ApplyFTSelection} function for each \textit{FTSelection}. Recall from the previous section that in the general case, the \textit{ApplyFTSelection} function takes in (a) the current evaluation context, (b) the current list of modifiers, (c) other information specific to each \textit{FTSelection}, and (d) the \textit{FullMatch} results of nested \textit{FTSelections}. Not all input parameters are used in every \textit{ApplyFTSelection} function corresponding to an \textit{FTSelection}, and for ease of exposition, we drop the irrelevant parameters when specifying the semantics of \textit{ApplyFTSelection} for each \textit{FTSelection}. The \textit{ApplyFTSelection} function always returns a \textit{FullMatch} for every \textit{FTSelection}.

We use an XQuery function to specify the semantics of each \textit{ApplyFTSelection}. This is possible because the inputs and output of the function can be specified in XML. Specifically, the evaluation context is already represented as a sequence of XQuery nodes. The list of modifiers can be represented as a sequence of modifier XML elements. A \textit{FullMatch} can be represented in XML form as described in Section 3.2.3.

The formal semantics of the \textit{ApplyFTSelection} function for each \textit{FTSelection} is specified in terms of four functions. How these four functions are computed is implementation-defined, but the functions have to satisfy some well-defined properties. We first present the properties of the implementation-defined functions, and then present the semantics of \textit{ApplyFTSelection} in terms of these functions.

\subsection{Implementation-defined functions}

The following four functions must be defined by any compliant TeXQuery implementation:

\begin{Verbatim}
function fts:matchStr($evaluationContext as Node*,
    $modifierContext as fts:ModifierContext,
    $searchToken as xs:string
  ) as fts:Position*
\end{Verbatim}

The above function returns all the positions in nodes in $evaluationContext that match the search token $searchToken when using the modifiers in $modifierContext. The modifiers that occur at the beginning of the list should be applied before modifiers that occur later in the list.

\begin{Verbatim}
function fts:posDistance($modifierContext as fts:ModifierContext,
    $pos1 as fts:Position,
    $pos2 as fts:Position
  ) as xs:integer
\end{Verbatim}

The above function returns the number of linguistic tokens that occur in positions between the positions $pos1 and $pos2. For example, two consecutive positions have a distance of 0. If $pos1
and $pos2$ are the same, then the distance is also defined to be 0. The $modifierContext$ may specify linguistic tokens that must be ignored in computing this distance (e.g., stop words, special characters in “without special characters” context, terms from ignored tags or element content).

function fts:paraDistance($modCtx as fts:ModifierContext,
   $p1 as fts:Position,
   $p2 as fts:Position
) as xs:integer

The above function returns the number of paragraphs that occur between the positions $pos1$ and $pos2$. The $modifierContext$ may specify linguistic tokens that must be ignored in computing this distance; specifically, paragraphs consisting entirely of ignored linguistic tokens are not counted when computing the distance.

function fts:sentenceDistance($modCtx as fts:ModifierContext,
   $p1 as fts:Position,
   $p2 as fts:Position
) as xs:integer

The above function returns the number of sentences that occur between the positions $pos1$ and $pos2$. The $modifierContext$ may specify linguistic tokens that must be ignored in computing this distance; specifically, paragraphs consisting entirely of ignored linguistic tokens are not counted when computing the distance.

We now specify the semantics of each ApplyFTSelection function in terms of the above functions.

5.2.2. Semantics of FTStringSelection

We only consider the case where FTStringSelection is a single search token. The other cases can be rewritten as complex FTSelections that operate on single string FTStringSelections, as described in Part I. The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, the search token, and the position where the search token occurs in the query. Since FTStringSelection does not have nested FTSelections, the ApplyFTSelection does not take in any FullMatch parameters corresponding to nested FTSelection results. The function definition is given below.

define function fts:ApplyFTSelection(
   $evaluationContext as Node*,
   $modifierContext as fts:ModifierContext,
   $searchToken as xs:string,
   $queryPos as xs:integer
) as element(fullMatch, fts:FullMatch) {

<fullMatch>
{let $token_pos := fts:matchStr($evaluationContext,
   $modifierContext,
   $searchToken)
for $pos in $token_pos
return
Intuitively, the FullMatch corresponding to an FTStringSelection corresponds to a set of SimpleMatches, each of which is associated with a position where the corresponding search token was found. For example, the FullMatch result for the FTStringSelection “Mustang” evaluated in the context of the sample document from Section 3 will be (in graphical terms):

### 5.2.3. FTOrConnective

The parameters of the ApplyFTSelection function are the two FullMatch parameters corresponding to the results of the two nested FTSelections. The evaluation context and the modifier context are not used in this case. The function definition is given below.

```plaintext
define function fts:ApplyFTSelection (  
    $fullMatch1 as element(fullMatch, fts:FullMatch),  
    $fullMatch2 as element(fullMatch, fts:FullMatch)  
) as element(fullMatch, fts:FullMatch){
    <$fullMatch>
        ($fullMatch1/simpleMatch
         $fullMatch2/simpleMatch)
    </$fullMatch>
}
The function creates a new `FullMatch` whose `SimpleMatches` are simply the union of those found in the input `FullMatches`. The rationale for this semantics is that each `SimpleMatch` represents one possible “solution” to the corresponding `FTSelection`. Thus, if we “or” two `FullMatches`, a `SimpleMatch` from either of the `FullMatches` should also be a solution.

As an example, consider the `FTSelection` “Mustang” || “Honda” in the context of the sample document in Section 3. The `FullMatches` corresponding to “Mustang” and “Honda” are:

```
<table>
<thead>
<tr>
<th>StrInclude</th>
<th>queryString: &quot;Mustang&quot;</th>
<th>queryPos: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos: 7</td>
<td>Pos: 45</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StrInclude</th>
<th>queryString: &quot;Honda&quot;</th>
<th>queryPos: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos: 24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The `FullMatch` produced by `ApplyFTSelection` for `FTOrConnective` is:

```
<table>
<thead>
<tr>
<th>StrInclude</th>
<th>queryString: &quot;Mustang&quot;</th>
<th>queryPos: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos: 7</td>
<td>Pos: 45</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StrInclude</th>
<th>queryString: &quot;Mustang&quot;</th>
<th>queryPos: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos: 45</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>StrInclude</th>
<th>queryString: &quot;Honda&quot;</th>
<th>queryPos: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pos: 24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
5.2.4. FTAndConnective

The parameters of the ApplyFTSelection function are the two FullMatch parameters corresponding to the results of the two nested FTSelections. The evaluation context and the modifier context are not used in this case. The function definition is given below.

```xml
define function fts:ApplyFTSelection (
  $fullMatch1 as element(fullMatch, fts:FullMatch),
  $fullMatch2 as element(fullMatch, fts:FullMatch))
  as element(fullMatch, fts:FullMatch){
  <fullMatch>
    {for $sm1 in $fullMatch1/simpleMatch
      for $sm2 in $fullMatch2/simpleMatch
        return
          <simpleMatch>
            {$sm1/*
            $sm2/*}
          <simpleMatch>
    </fullMatch>
  }
```

Intuitively, the result of a conjunction is a new FullMatch that contains the “Cartesian product” of the simple matches of the participating FTSelections. Every resulting simple match is formed the combination of the stringInclude components and stringExclude components from each of the FullMatches of the nested FTSelection conditions. Thus every simple match will contain the positions to satisfy a SimpleMatch from both original FTSelections and will exclude the positions that will violate the same SimpleMatches.

As an example let us consider the FTSelection "Mustang" && "rust" in the context of the sample document in 3. The source FullMatches are:
The *FTAndConnective* converts them to:

5.2.5. FTNegation

The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, and one *FullMatch* parameter corresponding to the result of the nested *FTSelections* to be negated. The evaluation context and the modifier context are not used in this case. The function definition is given below.

```javascript
define function fts:InvertStringMatch($strm) {
    if ($strm instanceof element(stringExclude)) {
```
The process of the generation of the resulting full match of an FTNegation resembles the transformation of a negation of prepositional formula in DNF back to DNF. The intuition is that negation of a FullMatch requires the inversion of all the conditions on the nodes encoded by the FullMatch.

In the implementation above, this inversion is implemented as follows. The function fts:invertStringMatch inverts a stringInclude into a stringExclude and vice versa. The function fts:neg_helper transforms the source SimpleMatches into the resulting SimpleMatches by combining a the inversions of a stringInclude or stringExclude component from every source SimpleMatch into a new SimpleMatch.

As an example, let us consider the FTSelection ! (“Mustang” || “Honda”) in the context of the sample document in 3. The source FullMatch is:
The \textit{FTNegation} will transform it to:

\textit{5.2.6. FTMildNegation}

The parameters of the \textit{ApplyFTSelection} function are the two \textit{FullMatch} parameters corresponding to the results of the two nested \textit{FTSelections}. The evaluation context and the modifier context are not used in this case. The function definition is given below.

\begin{verbatim}
define function fts:ApplyFTSelection ( 
    $fullMatch1 as element(fullMatch, fts:FullMatch), 
    $fullMatch2 as element(fullMatch, fts:FullMatch)) 
    as element(fullMatch, fts:FullMatch){ 
    <fullMatch> 
        {let $pos2=$fullMatch2/simpleMatch/stringInclude/docPos 
        return 
        $fullMatch1/simpleMatch[./stringInclude/docPos != $pos2] 
    }
\end{verbatim}
The resulting FullMatch consists of those SimpleMatches of the first operand that do not mention in their stringInclude components positions mentioned in a stringInclude component in the FullMatch of the second operand.

As an example, consider the FTSelection ("Ford" ignore "Ford Mustang") in the context of the sample document in 3. The source FullMatches are:

```
} 
</fullMatch>
```

```plaintext

```

<table>
<thead>
<tr>
<th>SimpleMatch</th>
<th>SimpleMatch</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrInclude</td>
<td>StrInclude</td>
</tr>
<tr>
<td>queryString: &quot;Ford&quot;</td>
<td>queryString: &quot;Ford&quot;</td>
</tr>
<tr>
<td>queryPos: 1</td>
<td>queryPos: 1</td>
</tr>
<tr>
<td>Pos: 6</td>
<td>Pos: 44</td>
</tr>
</tbody>
</table>

and

```

```

```

```plaintext

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```

```
The FTMildNegation transform these to an empty FullMatch because both position 6 and position 44 from the first FullMatch contain only positions from stringInclude components of the second FullMatch.

5.2.7. FTOOrderSelection

The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, and one FullMatch parameter corresponding to the result of the nested FTSelections. The evaluation context and the modifier context are not used in this case. The function definition is given below.

```xml
define function fts:ApplyFTSelection (  
    $fullMatch as element(fullMatch, fts:FullMatch))  
as element(fullMatch, fts:FullMatch){  
<fullMatch>  
{$fullMatch/simpleMatch[  
every $si1 in ./stringInclude,  
    $si2 in ./stringInclude  
satisfies  
    ($si1/docPos/@abs<=$si2/docPos/@abs and  
    $si1/@queryPos<=$si2/@queryPos)  
or  
    ($si1/docPos/@abs>=$si2/docPos/@abs and  
    $si1/@queryPos>=$si2/@queryPos)  
]  
}  
</fullMatch>
}
```

The resulting FullMatch contains all SimpleMatches of the parameter whose positions in the stringInclude elements are in the order of the query positions of their query strings.

As an example, consider the FTSelection (“great” && “condition”) in this order in the context of the sample document in 3. The source FullMatch is:
Continues on next diagram
The \textit{FTOrderSelection} will return only the second and the third part. The positions of terms in the first part are in reverse order to the query terms as determined by \textit{queryPos}.

\subsection*{5.2.8. FTScopeSelection}

The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, and one \textit{FullMatch} parameter corresponding to the result of the nested \textit{FTSelections}. The evaluation context and the modifier context are not used in this case. The functions definitions depending on the type of the scope (node, paragraph, sentence) and the scope predicate (same, different) are given below.

In the case of a scope “\textit{same node}”, the semantics is given by the XQuery function:

\begin{verbatim}
define function fts:ApplyFTSelection (  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  as element(fullMatch, fts:FullMatch){  
  <$fullMatch>  
  <$fullMatch/simpleMatch[  
    every $si in ./stringInclude  
    satisfies $si/pos/elem = ./stringInclude[1]/pos/elem]  
  ]  
  </$fullMatch>  
}
\end{verbatim}

The semantic for the scope “\textit{different node}” is given by the function:

\begin{verbatim}
define function fts:ApplyFTSelection (  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  as element(fullMatch, fts:FullMatch){  
  <$fullMatch>  
  <$fullMatch/simpleMatch[  
    every $sil in ./stringInclude,  
  ]  
  </$fullMatch>  
}
\end{verbatim}
The semantics for the case of sentence or paragraph scope is analogous. In case of “same sentence”, the semantics is given by:

```xml
define function fts:ApplyFTSelection (  
    $fullMatch as element(fullMatch, fts:FullMatch))
  as element(fullMatch, fts:FullMatch){
    <fullMatch>
      {$fullMatch/simpleMatch[every $si in ./stringInclude
          satisfies $si/pos/@sentence = ./stringInclude[1]/pos/@sentence]}
    </fullMatch>
  }
</fts:ApplyFTSelection>
```

Similarly, the semantics for “different sentence” is given by:

```xml
define function fts:ApplyFTSelection (  
    $fullMatch as element(fullMatch, fts:FullMatch))
  as element(fullMatch, fts:FullMatch){
    <fullMatch>
      {$fullMatch/simpleMatch[
          every $si1 in ./stringInclude,  
          $si2 in ./stringInclude  
          satisfies $si1=$si2  
          or  
          $si1/pos/@sentence != $si2/pos/@sentence  
        ]}
    </fullMatch>
  }
</fts:ApplyFTSelection>
```

In case of “same paragraph”, the semantics is given by:

```xml
define function fts:ApplyFTSelection (  
    $fullMatch as element(fullMatch, fts:FullMatch))
```
Finally, the semantics for “different paragraph” is given by:

```xml
define function fts:ApplyFTSelection (  $fullMatch as element(fullMatch, fts:FullMatch))  as element(fullMatch, fts:FullMatch){  <fullMatch>  ($fullMatch/simpleMatch[  every $si1 in ./stringInclude,  $si2 in ./stringInclude  satisfies $si1/pos/@para =  ./stringInclude[1]/pos/@para  or  $si1/pos/@para != $si2/pos/@para  ]  )  </fullMatch>  }
```

If for instance the type of the scope is “node”, the semantics is straightforward. For every `SimpleMatch` from the `FullMatch` of the operand, it filters those that contain string matches from `stringInclude` only in the same (different) element node. The cases for scope type paragraph or sentence are analogous.

As an example, consider the `FTSelection` ("Mustang" && “Honda”) same node in the context of the sample document in 3. The source `FullMatch` is:
The FTScopeSelection will convert this to an empty FullMatch because neither SimpleMatches contain positions from a single element.

5.2.9. FTDistanceSelection

The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, one FullMatch parameter corresponding to the result of the nested FTSelections, and one or two integers depending on the range specified FTRangeSpec used. The evaluation context is not used in this case, but the modifier context is needed because some of the linguistic tokens may need to be ignored (e.g. because they occur in the stop-words list) and therefore must not be counted against the distance. The semantics for the different cases depending on the distance units (words – linguistic tokens, paragraphs, sentences) and the FTRangeSpec used are given below.

The function for the case “word distance exactly N” is presented below:

```xml
define function fts:ApplyFTSelection (  
  $modCtx as fts:ModifierCtx,  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  $n as xs:integer)  
  as element(fullMatch, fts:FullMatch){
<fullMatch>
  (for $sm in $fullMatch/simpleMatch  
    let $sorted=  
      for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]  
        order by $si/docPos/@abs ascending  
        return $si  
      where every $index in (1 to fn:count($sorted)-1)  
        satisfies  
          fts:posDistance($SMCtx,  
                        $sorted[$index]/docPos,  
...
Similarly, the semantics for the case of “word distance at least $N$” is presented below:

```xml
define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
$\text{n as xs:integer)}  
    as element(fullMatch, fts:FullMatch){  
    <fullMatch>  
    {for $sm in $fullMatch/simpleMatch  
        let $\text{sorted}=$  
            for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]  
                order by $si/docPos/@abs ascending  
                return $si  
                where every $\text{index}$ in (1 to fn:count($\text{sorted})-1)  
                    satisfies  
                        fts:posDistance($\text{SMCtx},  
                            $\text{sorted}[$\text{index}]/docPos,  
                            $\text{sorted}[$\text{index}+1]/docPos) >= $\text{n}  
                    return $\text{sm}  
    }  
    </fullMatch>  
}
```

The semantics for the case of “word distance at most $N$” is given by:

```xml
define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
$\text{n as xs:integer)}  
    as element(fullMatch, fts:FullMatch){  
    <fullMatch>  
    {for $sm in $fullMatch/simpleMatch  
        let $\text{sorted}=$  
            for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]  
                order by $si/docPos/@abs ascending  
                return $si  
            where every $\text{index}$ in (1 to fn:count($\text{sorted})-1)  
                satisfies  
                    fts:posDistance($\text{SMCtx},  
                        $\text{sorted}[$\text{index}]/docPos,  
                        $\text{sorted}[$\text{index}+1]/docPos) <= $\text{n}  
                return $\text{sm}  
    }  
    </fullMatch>  
}
```
return $si
where every $index in (1 to fn:count($sorted)-1)
satisfies
fts:posDistance($SMCtx,
    $sorted[$index]/docPos,
    $sorted[$index + 1]/docPos) <= $n
return $sm
}
</fullMatch>
}
The semantics for the final case of "word distance from $M$ to $N$" is given by:

define function fts:ApplyFTSelection ( 
    $modCtx as fts:ModifierCtx,
    $fullMatch as element(fullMatch, fts:FullMatch))
    $m as xs:integer,
    $n as xs:integer)
as element(fullMatch, fts:FullMatch) {
<fullMatch>
    {for $sm in $fullMatch/simpleMatch
        let $sorted=
            for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
                order by $si/docPos/@abs ascending
                return $si
        where every $index in (1 to fn:count($sorted)-1)
            satisfies
                let $dist = fts:posDistance(
                    $SMCtx,
                    $sorted[$index]/docPos,
                    $sorted[$index + 1]/docPos)
                        return $m <= $dist and $dist <= $n
            return $sm
    }
</fullMatch>
}
The function for the case "sentence distance exactly $N$" is presented below:

define function fts:ApplyFTSelection ( 
    $modCtx as fts:ModifierCtx,
$fullMatch as element(fullMatch, fts:FullMatch)
$n as xs:integer)

as element(fullMatch, fts:FullMatch){

<fullMatch>
{for $sm in $fullMatch/simpleMatch

let $sorted= 
  for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
  order by $si/docPos/sentence ascending
  return $si

where every $index in (1 to fn:count($sorted)-1)
satisfies
  fts:sentenceDistance($SMCtx, 
    $sorted[$index]/docPos, 
    $sorted[$index + 1]/docPos) = $n

return $sm
}

</fullMatch>
}

Similarly, the semantics for the case of “sentence distance at least $N$” is presented below:

define function fts:ApplyFTSelection (
  $modCtx as fts:ModifierCtx, 
  $fullMatch as element(fullMatch, fts:FullMatch))
$n as xs:integer)

as element(fullMatch, fts:FullMatch){

<fullMatch>
{for $sm in $fullMatch/simpleMatch

let $sorted= 
  for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
  order by $si/docPos/sentence ascending
  return $si

where every $index in (1 to fn:count($sorted)-1)
satisfies
  fts:sentenceDistance($SMCtx, 
    $sorted[$index]/docPos, 
    $sorted[$index + 1]/docPos) >= $n

return $sm
}

}
The semantics for the case of “sentence distance at most $N$” is given by:

```xml
<fullMatch>
{for $sm in $fullMatch/simpleMatch
 let $sorted=
  for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
   order by $si/docPos/sentence ascending
   return $si
 where every $index in (1 to fn:count($sorted)-1)
   satisfies
    fts:sentenceDistance($SMCtx,
     $sorted[$index]/docPos,
     $sorted[$index + 1]/docPos) <= $n
  return $sm
}
</fullMatch>
```

The semantics for the final case of “sentence distance from $M$ to $N$” is given by:

```xml
<fullMatch>
{for $sm in $fullMatch/simpleMatch
 let $sorted=
  for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
   order by $si/docPos/sentence ascending
   return $si
 where every $index in (1 to fn:count($sorted)-1)
   satisfies
    fts:sentenceDistance($SMCtx,
     $sorted[$index]/docPos,
     $sorted[$index + 1]/docPos) <= $n
  return $sm
}
</fullMatch>
```
satisfies
let $dist = fts:sentenceDistance(
    $SMCtx,
    $sorted[$index]/docPos,
    $sorted[$index + 1]/docPos)
return $m <= $dist and $dist <= $n
return $sm
</fullMatch>

The function for the case “paragraph distance exactly $N$” is presented below:

define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
    $n as xs:integer)
as element(fullMatch, fts:FullMatch){
    <fullMatch>
        {for $sm in $fullMatch/simpleMatch
            let $sorted=
                for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]
                    order by $si/docPos/para ascending
                    return $si
            where every $index in (1 to fn:count($sorted)-1)
                satisfies
                    fts:paraDistance($SMCtx,
                        $sorted[$index]/docPos,
                        $sorted[$index + 1]/docPos) = $n
            return $sm
        }
    </fullMatch>
}

Similarly, the semantics for the case of “paragraph distance at least $N$” is presented below:

define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
    $n as xs:integer)
as element(fullMatch, fts:FullMatch){

The semantics for the case of “paragraph distance at most $N$” is given by:

\[
\text{define function } \texttt{fts:ApplyFTSelection (}
\text{ $modCtx$ as \texttt{fts:ModifierCtx,}}
\text{ $fullMatch$ as element(\texttt{fullMatch, fts:FullMatch}))}
\text{ $n$ as \texttt{xs:integer)}}
\text{ as element(\texttt{fullMatch, fts:FullMatch})}{
\begin{align*}
\text{<fullMatch>}
\text{(for } $sm$ \text{ in } \texttt{fullMatch/simpleMatch}}
\text{ let } $\text{sorted}= \\
\text{ for } $si$ \text{ in } ./\texttt{stringInclude[isValidPos($SMCtx, ./pos)]}}
\text{ order by } $si/docPos/para \text{ ascending} \\
\text{ return } $si
\text{ where every } $index \text{ in } (1 \text{ to } fn:count($sorted)-1) \\
\text{ satisfies} \\
\texttt{fts:paraDistance($SMCtx,}}
\text{ $\text{sorted[$index]}/docPos,}$
\text{ $\text{sorted[$index + 1]}/docPos) \geq n}$
\text{ return } $sm$
\text{ )}
\text{ </fullMatch>}
\end{align*}
\]

The semantics for the final case of “paragraph distance from $M$ to $N$” is given by:
define function fts:ApplyFTSelection (  
  $modCtx as fts:ModifierCtx,  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  $m as xs:integer,  
  $n as xs:integer)  
  as element(fullMatch, fts:FullMatch){  
  <fullMatch>  
  {for $sm in $fullMatch/simpleMatch  
  let $sorted=  
    for $si in ./stringInclude[isValidPos($SMCtx, ./pos)]  
    order by $si/docPos/para ascending  
    return $si  
  where every $index in (1 to fn:count($sorted)-1)  
  satisfies  
    let $dist = fts:paraDistance(  
      $SMCtx,  
      $sorted[$index]/docPos,  
      $sorted[$index + 1]/docPos)  
    return $m <= $dist and $dist <= $n  
  return $sm  
  }  
  </fullMatch>  
}  

Intuitively, the resulting FullMatch contains those SimpleMatches of the operand that satisfy the condition that the distance (measured in words/linguistic tokens, sentences, or paragraphs) for every couple of consecutive valid positions in stringInclude elements is in the specified interval. Here by consecutive, we mean with no other valid positions from the same stringInclude element between them.

As an example, consider the FTDistanceSelection ("Ford Mustant" && "excellent") word distance at most 3 over the sample document fragment in 3. The four simple matches of the source full match for ("Ford Mustant" && "excellent") are given below:
The result for the above \textit{FTDistanceSelection} will consist of only the first simple match because only their the distance between consecutive positions (0 and 2 in this case) are less or equal to 3.

\subsection*{5.2.10. FTWindowSelection}

The parameters of the \textit{ApplyFTSelection} function are the evaluation context, the list of context modifiers, one \textit{FullMatch} parameter corresponding to the result of the nested \textit{FTSelections}, and one or two integers depending on the range specified \textit{FTRangeSpec} used. The evaluation context is not used in this case, but the modifier context is needed because some of the linguistic tokens may need to be ignored (e.g. because they occur in the stop-words list) and therefore must not be counted against the window size. The semantics for the different cases depending on the range specification \textit{FTRangeSpec} used follow.

The function for the case “window exactly \( N \)” is presented below:

```plaintext
define function fts:ApplyFTSelection (  $modCtx as fts:ModifierCtx,
```
The function for the case “window at least $N$” is presented below:

```xml
define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
    $n as xs:integer)  
as element(fullMatch, fts:FullMatch){
    <fullMatch>
        {for $sm in $fullMatch/simpleMatch
            let $pos := $sm/docPos[isValidPos($SMCtx, .)]
            let $max_pos = fn:max($pos/@abs)
            let $min_pos = fn:min($pos/@abs)
            let $window := fts:posDistance($min_pos, $max_pos) + 2
            where $window >= $n
            return $sm
        }
    </fullMatch>
}
```

The function for the case “window at least $N$” is presented below:

```xml
define function fts:ApplyFTSelection (  
    $modCtx as fts:ModifierCtx,  
    $fullMatch as element(fullMatch, fts:FullMatch))  
    $n as xs:integer)  
as element(fullMatch, fts:FullMatch){
    <fullMatch>
        {for $sm in $fullMatch/simpleMatch
            let $pos := $sm/docPos[isValidPos($SMCtx, .)]
            let $max_pos = fn:max($pos/@abs)
            let $min_pos = fn:min($pos/@abs)
            let $window := fts:posDistance($min_pos, $max_pos) + 2
            where $window >= $n
            return $sm
        }
    </fullMatch>
}
```
The function for the case “window from M to N” is presented below:

```xml
<fullMatch>
  {for $sm in $fullMatch/simpleMatch
    let $pos := $sm/docPos[isValidPos($SMCtx, .)]
    let $max_pos = fn:max($pos/@abs)
    let $min_pos = fn:min($pos/@abs)
    let $window := fts:posDistance($min_pos, $max_pos) + 2
    where $window <= $n
      return $sm
  }
</fullMatch>
```

Intuitively, the resulting FullMatch contains those SimpleMatches of the operand that satisfy the condition that the distance between the maximum position and the minimum position plus two (because the include both positions) is within the specified interval.

As an example, consider the FTWindowSelection (“Ford Mustant” && “excellent”) word distance at most 20 over the sample document fragment in 3. The four simple matches of the source full match for (“Ford Mustant” && “excellent”) are given below:
The result for the above \textit{FTWindowSelection} will consist of the first two simple matches because their window sizes are 5 and 19 respectively.
5.2.11. FTTimesSelection

The parameters of the ApplyFTSelection function are the evaluation context, the list of context modifiers, one \textit{FullMatch} parameter corresponding to the result of the nested \textit{FTSelections}, and one or two integers depending on the range specified \textit{FTRangeSpec} used. The evaluation context and the modifier context are not used in this case.

The function definitions, depending on the range specification \textit{FTRangeSpec} limiting the number of occurrences, follow.

```xml
define function fts:FormCombinations($sms, $times) {
    if (fn:count($sms) eq 0) then ()
    else if ($times eq 0) then ()
    else {
        fts:formCombination(fn:subsequence($sms, 2), $times)
        (<simpleMatch>
            $sms[1]
            fts:formCombinations(fn:subsequence($sms, 2),$times-1)/*
        </simpleMatch>
    }
}

define function fts::FormRange($sms, $l, $u) {
    let $lower_match :=
    <fullMatch>
        {fts:formCombinations($sms, $l) } 
    </fullMatch>
    return
    if ($l > $u) then ()
    else fts:applyAndConnective(
        <fullMatch>
            {fts:FormCombinations($sms, $l})
        </fullMatch>,
        fts::applyNegation(
            <fullMatch>
                {fts:FormCombinations($sms, $u+1)}
            </fullMatch>)
    )
}
```

We now define the semantics for the case “exactly $N$ occurrences”:

```xml
define function fts:ApplyFTSelection (
We next define the semantics for the case “at least \(N\) occurrences”:

```xml
define function fts:ApplyFTSelection (  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  $n as xs:integer)  
  as element(fullMatch, fts:FullMatch){  
  fts:formCombinations($fullMatch/simpleMatch, $n)  
}
```

We next define the semantics for the case “at most \(N\) occurrences”:

```xml
define function fts:ApplyFTSelection (  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  $n as xs:integer)  
  as element(fullMatch, fts:FullMatch){  
  fts:formRange($fullMatch/simpleMatch, 0, $n)  
}
```

Finally, we define the semantics for the case “from \(M\) to \(N\) occurrences”:

```xml
define function fts:ApplyFTSelection (  
  $fullMatch as element(fullMatch, fts:FullMatch))  
  $m as xs:integer,  
  $n as xs:integer)  
  as element(fullMatch, fts:FullMatch){  
  fts:formRange($fullMatch/simpleMatch, $m, $n)  
}
```

The intuition is as follows. The way to ensure that there are at least \(N\) different matches of an \(FTSelection\) is to ensure that at least \(N\) of its \(SimpleMatches\) occur simultaneously. This is similar to forming their conjunction: combine \(N\) distinct simple matches into one simple match. Therefore, the full match for the selection condition involving the range specifier “at least \(N\)” is to form all possible combinations of \(N\) simple matches of the operand and form one simple match for each combination negating the rest of the simple matches. This operations is performed in the function fts:FormCombinations.

In the case of another range \([l, u]\), it is treated as the condition “at least \(l\) and not at least \(u+1\)”. This transformation is performed in the function fts:FormRange.

As an example, consider the \(FTTimesSelection\) “Mustang” at least 2 occurrences over the sample document fragment in 3. The source full match of the \(FTStringSelection\) “Mustang” is:
The result will consist of all couples of simple matches from above:
6. Example

We will now show the evaluation of a more elaborate example of FTSelection. We use the same sample document as in Section 3. For convenience, we present it again here.

<offer id="1000" price="10000">
    Ford Mustang 2000, 65K, excellent condition, runs great, AC, CC, power all
</offer>

<offer id="1001" price="8000">
    Honda Accord 1999, 78K, A/C, cruise control, runs and looks great, excellent condition
</offer>

<offer id="1005" price="5500">
    Ford Mustang, 1995, 150K highway mileage, little rust, excellent condition
</offer>

We will walk through the evaluation of the following FTSelection:

```
( ("mustang" &&
    ( ("great" | | "excellent") at least 2 occurrences)
  ) window at most 30
  &&
  ! "rust"
) same node
```
Step 1: Evaluate the \textit{FTStringSelection} "Mustang"

```
Step 1: Evaluate the \textit{FTStringSelection} "Mustang"

FullMatch

SimpleMatch

StrInclude
queryPos:1
queryStr:"Mustang"
Pos: 7

SimpleMatch

StrInclude
queryPos:1
queryStr:"Mustang"
Pos: 45
```

Step 2: Evaluate the \textit{FTStringSelection} "great"

```
Step 2: Evaluate the \textit{FTStringSelection} "great"

FullMatch

SimpleMatch

StrInclude
queryPos:2
queryStr:"great"
Pos: 13

SimpleMatch

StrInclude
queryPos:2
queryStr:"great"
Pos: 35
```
Step 3: Evaluate the FTStringSelection “excellent”

Step 4 - Apply the FTOrSelection (“great” || “excellent”): form the union of the SimpleMatch’es
Step 5 - Apply the \textit{FTTimesSelection} ("great" \text{ || } "excellent") at least 2 occurrences: form 2-tuples (couples) of \textit{SimpleMatch}'es

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{diagram.png}
\end{figure}

\textit{Continues on next page}
Step 6 - Apply the FTAndConnective “Mustang” && (“great” || “excellent”) at least 2 occurrences): form the “Cartesian product” of SimpleMatch’es

Continues on next page
FullMatch

SimpleMatch

StrInclude
queryPos:2
queryStr:"great"
Pos: 13

StrInclude
queryPos:1
queryStr:"Mustang"
Pos: 7

StrInclude
queryPos:3
queryStr:"excellent"
Pos: 36

SimpleMatch

StrInclude
queryPos:1
queryStr:"Mustang"
Pos: 45

StrInclude
queryPos:2
queryStr:"great"
Pos: 13

StrInclude
queryPos:3
queryStr:"excellent"
Pos: 62

SimpleMatch

StrInclude
queryPos:2
queryStr:"great"
Pos: 13

StrInclude
queryPos:1
queryStr:"Mustang"
Pos: 7

StrInclude
queryPos:3
queryStr:"excellent"
Pos: 62

Continues on next page
Continues on next page
Step 7 - Apply the FTWindowSelection ("Mustang" && ("great" || "excellent") at least 2 occurrences)) window at most 30: filter out SimpleMatch’es for which the window is not less than or equal to 30

Continues on next page
Step 8 - Match \textit{FTStringSelection} “rust”

Step 9 - Apply \textit{FTNegation} ! “rust”: transform the \textit{StringInclude} into \textit{StringExclude}
Step 10 - Apply the \textit{FTAndConnective} \((\text{"Mustang"} \&\& ((\text{"great"} || \text{"excellent"}) at least 2 occurrences)) \) window at most 30) \&\& ! \text{"rust"}: form the \text{"Cartesian product"} of the \textit{SimpleMatch}'es

\text{Continues on next page}
Continues on next page
Step 11: Apply the final *FTScopeSelection* – filter out *SimpleMatch*’es whose positions are not within the same node

This is the final *FullMatch*!
7. References

