Project Description

XPLite

In this section we describe XPLite, a path expression language you need to implement.

The syntax of XPLite is described using the following grammar:

$$
\begin{aligned}
\text{Expression} &::= (\text{'/'LocationStep})^* \\
\text{LocationStep} &::= \text{Axis`::'} \text{ NodeTest}[\text{`['Predicate`]'}]^* \\
\text{Axis} &::= \texttt{self} \mid \\
&\quad \texttt{parent} \mid \\
&\quad \texttt{child} \mid \\
&\quad \texttt{attribute} \mid \\
&\quad \texttt{ancestor} \mid \\
&\quad \texttt{descendant} \mid \\
&\quad \texttt{following} \mid \\
&\quad \texttt{preceding} \mid \\
&\quad \texttt{following-sibling} \mid \\
&\quad \texttt{preceding-sibling} \\
\text{NodeTest} &::= NodeName \mid \\
&\quad \texttt{*} \mid \\
&\quad \texttt{node()} \mid \\
&\quad \texttt{attribute()} \mid \\
&\quad \texttt{text()}
\end{aligned}
$$

Predicate ::= LocationStep′/′LocationStep*|
PredicateExpression OP PredicateExpression

PredicateExpression ::= function|
ConstantValue

function ::= `position()`|
`string()`|
`last()`

OP ::= = | < | > | >= | <= | <>

Informally, XPLite is a (n almost a) subset of XPath that contains all its major axes axes (self, child, parent, descendant, ancestor, following, preceding, following-sibling and preceding-sibling and attribute), five types of node tests, and a simplified version of predicates.

In particular, each individual predicate is either an XPLite expression or a comparison between values of built-in functions and constants (you can compare a value of one function to a value of another function, as well as a value of a function to a constant; you can also compare constants, but this is less useful). Each location step can contain multiple predicates.

The semantics of XPLite is defined as follows.

- All expressions, except the expressions in the predicates are considered to be absolute, i.e., they start from the root of the XML repository specified in the command.
- As a special case, XPLite query of the form / returns the root of the repository, that is, the content of the entire XML repository[1].
- The XML tree on which XPLite expressions are evaluated has the following types of nodes:
  - root. The root of the repository is the only node of this type.
  - element. A node that represents an XML element.
  - text. A node that contains text content only.
  - attribute. A node that contains information about an attribute.
- The semantics of each **location step** is the same as in XPath: it is treated as a trasnformation of the input node set into the output node set based on the semantics of the individual components of the location step.

[1]XPLite queries will be directly preceeded by the repository name: `RETURN document("RepositoryName")/` is the command to retrieve the root of the given repository

- The semantics of the **axes** follows XPath. To simplify things, we explicitely assume that nodes whose content is #PCDATA have no children or descendants.

- Note that `attribute` nodes are not reachable via `child` or `descendant` axes. They are reachable only via the `attribute` axis. However, if the context node is an attribute node, the element node to which it belongs is reachable from it via `parent` axis.

  ```
  /descendant::foo/attribute::bar/parent::node()
  ```

  returns all foo nodes that have an attribute bar.

- The semantics of the **node tests** follows XPath XPath. The "*" nodetest is the abbreviation for the `node()` nodetest[2]. `text()` nodetest is satisfied by XML element nodes which have no descendants, but do have non-trivial content[3].

  For example, in the following document,

  ```
  <root>
       <a>This is a</a>
       <b>test</b>
  </root>
  ```

  element `<root>` has two children: `<a>` and `<b>`. Both satisfy the `text()` nodetest.

- The semantics of the **predicates** is as follows.

  - If a location step contains more than one predicate then, a node is added to the output node set iff all predicates are true on it (i.e., we use conjunction).
  - Predicates of type XPLite Expression are considered to be **relative** and are evaluated for each of the nodes in the input nodeset (context). They are evaluated to true iff XPLite expression yields a non-empty nodeset as the answer, otherwise, they evaluate to false.
  - Predicates of the form $function$ OP $value$ are evaluated as follows. The function is evaluated on the context node, and its value is compared using the specified operator (the semantics of all operators is traditional) to the specified constant value. If the relationship between the function value and the constant is true, the predicate evaluates to true, otherwise, it evaluates to false.

[2] This is the only abbreviation we allow in XPLite syntax

[3] I.e., those XML element nodes that have a record in the `ContentIndex` structure.

– function `position()` returns the position of the current node in its current context - i.e., in the current node set. For example,

```
/child::chapter[position()=4]
```

returns the fourth `chapter` child of the root. At the same time,

```
/descendant::f/parent::d[position()=2]
```

looks for the node `<d>` which has a child `<f>` and which is the second such node in the document order. In the following example:

```
<root>
    <d><f>1</f></d>
    <d><g>2</g></d>
    <f><d>3</d></f>
    <d><f>4</f></d>
</root>
```

the abovementioned query will retrieve in the last element `<d>` (`<d><f>4</f></d>`).

The context is *always ordered in* ***document order***, i.e., the order of appearance of the starting tag of each node in the XML document.

– function `string()` has dual meaning. On nodes of type `text` (i.e., those that pass `text()` node test), this function returns the #PCDATA content of the node. On nodes of type `attribute` it returns the value of the current attribute. On all other nodes it returns empty string. The result of this function must be compared to a string value.

– function `last()` returns the size of the current context, i.e. the number of nodes in the context nodeset (nodelist). Using the XML document from above, in the following expression

```
/descendant::f[last()=3]
```

`last()` will evaluate to 3 – there are three nodes `<f>` found in the document, they form the node set that `last()` gets as the input.

At the same time,

```
/descendant::f[position()=last()]
```

returns only the last `<f>` node from the document (`<f>4</f>`).