Semantic Characterizations of XPath

Maarten Marx

Informatics Institute,
University of Amsterdam, The Netherlands

CWI, April, 2004
Overview

- **Navigational** XPath is a language to specify sets and paths in node-labelled sibling ordered trees.

- XPath 1.0 is a sublanguage of first order logic with predicates
  - descendant
  - right sibling
  - unary predicates for the node labels

- **This talk:** exactly which sublanguage?

- This talk has two dimensions:
  1. defining answer sets vs defining paths.
  2. XPath 1.0 vs Conditional XPath.
Syntax of navigational XPath 1.0

\[
\text{locpath} ::= \text{axis ‘::’ ntst} \mid \text{axis ‘::’ ntst ‘[fexpr]’} \mid \\
\quad \text{‘/’ locpath} \mid \text{locpath ‘/’ locpath} \mid \text{locpath ‘|’ locpath} \\
\text{fexpr} ::= \text{locpath} \mid \text{not fexpr} \mid \text{fexpr and fexpr} \mid \text{fexpr or fexpr} \\
\text{axis} ::= \text{self} \mid \text{child} \mid \text{parent} \mid \\
\quad \text{descendant} \mid \text{descendant_or_self} \mid \\
\quad \text{ancestor} \mid \text{ancestor_or_self} \mid \\
\quad \text{following_sibling} \mid \text{preceding_sibling} \mid \text{following} \mid \text{preceding},
\]

• where “locpath” (pronounced as *location path*) is the start production,

• “axis” denotes axis relations

• “ntst” denotes tags labeling document nodes or the star ‘*’

• The “fexpr” are called *filter expressions* after their use as filters in location paths.
The models of navigational XPath 1.0

The semantics of XPath expressions is given with respect to an XML document modeled as a finite node labeled sibling ordered tree 
\((N, R_{\downarrow}, R_{\Rightarrow}, P_i)_{i \in \Sigma}\).

- \(N\) is the domain (the set of nodes)
- \(R_{\downarrow}\) is descendant
- \(R_{\Rightarrow}\) is following-sibling
- the \(P_i\) are subsets of \(N\).

We will also speak about such trees in First Order Logic in this signature.
Semantics of navigational XPath 1.0

\[
\begin{align*}
[\text{axis} :: t]_M &= \{(n, n') | n[\text{axis} ]_M n' \text{ and } t(n')\} \\
[\text{axis} :: t[e]]_M &= \{(n, n') | n[\text{axis} ]_M n' \text{ and } t(n') \text{ and } E_M(n', e)\} \\
[/\text{locpath}]_M &= \{(n, n') | (\text{root}, n') \in [\text{locpath} ]_M\} \\
[\text{locpath}/\text{locpath}]_M &= [\text{locpath} ]_M \circ [\text{locpath} ]_M \\
[\text{locpath} | \text{locpath}]_M &= [\text{locpath} ]_M \cup [\text{locpath} ]_M \\
[\text{self}]_M &= \{(x, y) | x = y\} \\
[\text{descendant}]_M &= R_{\downarrow} \\
[\text{following_sibling}]_M &= R_{\Rightarrow} \\
[\text{child}]_M &= R_{\downarrow} \cap R_{\downarrow} \circ R_{\downarrow} \\
[\text{ancestor}]_M &= \text{descendant} ]_M^{-1} \\
\text{etc}
\end{align*}
\]

\[E_M(n, \text{locpath}) = \text{true} \iff \exists n' : (n, n') \in [\text{locpath} ]_M\]
\[E_M(n, \text{fexpr}_1 \text{ and } \text{fexpr}_2) = \text{true} \iff E_M(n, \text{fexpr}_1) = \text{true} \text{ and } E_M(n, \text{fexpr}_2) = \text{true}\]
\[E_M(n, \text{fexpr}_1 \text{ or } \text{fexpr}_2) = \text{true} \iff E_M(n, \text{fexpr}_1) = \text{true} \text{ or } E_M(n, \text{fexpr}_2) = \text{true}\]
\[E_M(n, \text{not } \text{fexpr}) = \text{true} \iff E_M(n, \text{fexpr}) = \text{false}\]
The paths of XPath 1.0

- Each XPath expression denotes a set of paths.
- We can think of the filter expressions as denoting a set of nodes.
- We first look at the expressive power of XPath in terms of defining paths.
The connectives of XPath

- What are the connectives of XPath?
The connectives of XPath

• What are the connectives of XPath?

• Clearly there is composition (’/’) and union (’ | ’) of paths.

• Then there is composition with a filter expression (’[F]’).

• And inside the filter expressions all boolean connectives are allowed.

• It turns out that this rather messy set can be streamlined.
The connectives of XPath II

Consider the following definition of path formulas:

\[ R ::= \text{axis} \mid ?A_i \mid R/R \mid R \mid R \mid \sim R, \]  

for axis one of Core XPath’s axis relations, for \( A_i \) a tagname, and the following meaning for the two new connectives:

\[
\begin{align*}
\llbracket ?A_i \rrbracket_M &= \{(x, x) \mid \text{the tag of } x \text{ is } A_i\} \\
\llbracket \sim R \rrbracket_M &= \{(x, y) \mid x = y \text{ and } \neg \exists z \ x \llbracket R \rrbracket_M z\}.
\end{align*}
\]

- **Example** \( \sim \text{child} \) defines the identical relation consisting of leaves, and \( \sim \text{parent} \) the singleton \( \{(\text{root}, \text{root})\} \).

- These are exactly the connectives defining the first order safe for bisimulation relations.
The connectives of XPath III

- **Theorem** The above definition is equivalent to the one of XPath 1.0.

- **Proof** Because $\sim R \equiv \text{self :: } *[\text{not } R]$ every relation defined above can be expressed as an XPath formula.

- For the other direction,
  - (Atoms) $\text{axis :: } P_i \equiv \text{axis} / ?P_i$ and $\text{axis :: } * \equiv \text{axis} / (?P \cup \sim ?P)$
  - Both languages contain union and composition.
  - Filter expressions are definable as follows

\[
\begin{align*}
? (\text{axis :: } A) & = \sim\sim (\text{axis} / ?A) \\
? (\text{axis :: } A[F]) & = \sim\sim (\text{axis} / ?A / ?F) \\
? (\text{not } A) & = \sim ?A \\
? (A \text{ and } B) & = ?A / ?B \\
? (A \text{ or } B) & = ?A \mid ?B.
\end{align*}
\]
Structural properties of XPath

• Benedikt, Fan and Kuper (ICDT 2003) gave an in depth analysis of a number of structural properties of positive vertical fragments of XPath.

• All their fragments allowing filter expressions are closed under intersection, while none is closed under complementation.

• This is also true for full XPath.
XPath is closed under intersection

• **Theorem** For every two XPath expressions $A$, $B$, there exists an XPath expression $C$ such that on every model $\mathcal{M}$, $[A]_\mathcal{M} \cap [B]_\mathcal{M} = [C]_\mathcal{M}$. 
XPath is closed under intersection

Proofsketch:

- Take $A, B$, distribute $\cup$ over $/$ and $\cup$ over $\cap$.
- So we just have to find the intersection of two composition formulas.
- Rewrite each into a normal form: being a union of forms
  - self, down, up, up*/right / down*, up*/left / down*.
- Now everything reduces to intersections of compositions in the same direction on the line.
- That is easy:

\[
\text{descendant} :: A/\text{descendant} :: B \cap \\
\text{descendant} :: C/\text{descendant} :: D \equiv \\
\text{descendant} :: A/\text{descendant} :: C/\text{descendant} :: B[\text{self} :: D] \\
\quad \cup \\
\text{descendant} :: C/\text{descendant} :: A/\text{descendant} :: B[\text{self} :: D].
\]
XPath is not closed under complement

• Suppose it was. We will derive a contradiction.
• Then the transitive closure of the relation child :: A would be expressible as

\[
\text{descendant}/?A \cap \text{descendant}/\sim?A/\text{descendant}. \quad (2)
\]

• Abbreviate this relation by \( R \). Then \( xRy \) holds iff

\[
x \text{ descendant } y \land A(y) \land \forall z((x \text{ descendant } z \land z \text{ descendant } y) \rightarrow A(z)).
\]

• Consider the query for all \( x \) s.t. \( \exists y(xRy \land \text{leaf}(y)) \).
• A standard argument shows that this set cannot be specified using less then three variables.
• This contradicts the Theorem below which states that the answer set of every XPath expression is equivalent to a first order formula in two variables.
Paths and answer sets

- From W3C Recommendation: The primary purpose of XPath is to address parts of an XML document.

- Each XPath path wff denotes a set of paths.

- Each XPath path wff also selects a set of nodes!

- Definition Given a tree $M$, and a path wff $R$,

  \[ \text{answer}_M(R) = \text{the range of } R = \{ n \mid \exists n' : n' R^m n \} \]

- We now measure the expressive power of XPath in defining answer sets.
Path wffs and filter wffs

• Compare the expressive power of path and filter wffs.

• Path wff $\textit{descendant} :: A$ selects the same set as the filter wff $A$ and $\textit{ancestor} :: *$.

• Observation Path wffs and filter wffs are equally expressive.

• These compare like procedural versus declarative specifications.
Path wffs and filter wffs

• **Corollary** (M., de Rijke 2004) XPath 1.0 is equally expressive as modal logic with modalities for

  child, descendant, following_sibling, and their inverses.

  (In fact this is part of the modal logic of trees of Blackburn, Meijer–Viol, de Rijke 95.)

• **Corollary** The answer set of any Core XPath expression can be defined by a $FO_2[\text{descendant, child, followingSibling}]$ formula in one free variable.
Expressive completeness

- The converse also holds!

- **Theorem** (M., de Rijke 2004) Every set of nodes defined by a $FO_2[\text{descendant, child, following\_sibling}]$ formula in one free variable is equivalent to a XPath filter expression.
Expressive completeness (Proofsketch)

- Let $\phi(x)$ be the first order formula.
- We give a recursive translation procedure due to Etessmai, Vardi, Wilke 97 for linear structures.
- $P_i(x)$ goes to self :: $p_i$ and the translation commutes with the booleans.
- Now consider $\exists y \phi(x, y)$. Then

$$
\phi(x, y) = \beta(R_0(x, y), \ldots, R_{r-1}(x, y), X_0(x), \ldots, X_{s-1}(x), Y_0(y), \ldots, Y_{s-1}(y)),
\text{atomic relations atoms, or existential wffs atoms, or existential wffs).}
$$

, with $\beta$ a boolean formula.
- By distribution we may assume that $\beta$ is a conjunction and the negations are in front of the atoms or existentials.
- Now bring the $X_i$ out of the scope of $\exists y$. They are of lower quantifier depth, whence can be translated.
- $\phi(x, y)$ is almost a modal formula!
Two variable property continued

\[ \phi(x, y) = \bigwedge (R_0(x, y), \ldots, R_{r-1}(x, y), Y_0(y), \ldots, Y_{s-1}(y)). \]

- If we could just replace that conjunction \( R_0(x, y), \ldots, R_{r-1}(x, y) \) by one relation we are done.
- But we can, because two points in a tree are related iff they stand in one of the following relations. Then each other relation becomes either true or false.

<table>
<thead>
<tr>
<th>( \tau(x, y) )</th>
<th>( \exists y(\tau(x, y) \land A(y)) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x = y )</td>
<td>self :: ( * [A'] )</td>
</tr>
<tr>
<td>( x R_\downarrow y )</td>
<td>child :: ( * [A'] )</td>
</tr>
<tr>
<td>( y R_\downarrow x )</td>
<td>parent :: ( * [A'] )</td>
</tr>
<tr>
<td>( x R_+ y )</td>
<td>following_sibling :: ( * [A'] )</td>
</tr>
<tr>
<td>( y R_+ x )</td>
<td>preceding_sibling :: ( * [A'] )</td>
</tr>
<tr>
<td>( x R_\downarrow y \land \neg x R_\downarrow y )</td>
<td>child :: */descendant :: ( * [A'] )</td>
</tr>
<tr>
<td>( y R_\downarrow x \land \neg y R_\downarrow x )</td>
<td>parent :: */ancestor :: ( * [A'] ).</td>
</tr>
</tbody>
</table>
Conditional XPath

- Natural extension of XPath 1.0.
- Allow transitive closure of `one_step_axis?test`.
- For instance,

  all nodes $n$ from which there exists a path to a leaf consisting just of $A$ nodes.

  is expressible as

  $$(\downarrow?A)^+ ?(\text{not dom(\downarrow)}).$$

- In conditional XPath we can express *until*. 
Until

- *Until* takes two arguments:
  1. a test at the final point;
  2. a test or instruction which is evaluated at all points *between* now and the final test point.
Until

- *Until* takes two arguments:
  1. a test at the final point;
  2. a test or instruction which is evaluated at all points *between* now and the final test point.

![Diagram with labeled points and arrows showing the sequence of points from n to n']

if and only if
Until

- *Until* takes two arguments:
  1. a test at the final point;
  2. a test or instruction which is evaluated at all points *between* now and the final test point.

\[
\text{Until}(A, B)
\]

\[
\begin{array}{c}
A \\
\end{array}\]

\[
\begin{array}{c}
\text{if and only if} \\
A
\end{array}\]

\[
\begin{array}{c}
\text{and} \\
A
\end{array}\]
Until

- *Until* takes two arguments:
  1. a test at the final point;
  2. a test or instruction which is evaluated at all points *between* now and the final test point.

\[
\text{Until}(A, B) \iff A_{n'} \land B_{n'}
\]
Conditional XPath

Theorem (M. 2004) Conditional XPath is equally expressive as first order logic with descendant and following-sibling.

This holds both for expressing answer sets and for expressing sets of paths.