XML Data Management

Part 12: Core XPath

Werner Nutt

Faculty of Computer Science
Master of Science in Computer Science

A.Y. 2012/2013

Freie Universität Bozen
Libera Università di Bolzano
Free University of Bozen · Bolzano

A Logic View of Databases

A database has two parts: **schema** and **instance**

The schema describes *how data is organized:*

- relations with their names and with number, names and types of attributes
- Example: $\mathtt{Person}(\mathtt{name}, \mathtt{gender}, \mathtt{age})$, $\mathtt{HasChild}(\mathtt{parent}, \mathtt{child})$

The instance contains the *actual data:*

- for every relation, there is a set of atoms complying with the schema,
- Example: $\{\mathtt{Person}(\mathtt{fred}, \mathtt{'male'}, 25), \mathtt{Person}(\mathtt{mary}, \mathtt{'female'}, 60),$
  $\mathtt{Person}(\mathtt{clara}, \mathtt{'female'}, 3), \mathtt{Person}(\mathtt{paul}, \mathtt{'male'}, 1),$
  $\mathtt{HasChild}(\mathtt{mary}, \mathtt{fred}), \mathtt{HasChild}(\mathtt{fred}, \mathtt{clara}),$
  $\mathtt{HasChild}(\mathtt{fred}, \mathtt{paul})\}$

Often, we ignore types and sometimes, we ignore also the attribute names

First-Order Queries

Definition

A **first-order query** has the form

$$Q = \{(x_1, \ldots, x_n) \mid \phi\}$$

where

- ϕ is a predicate logic formula
- $x_1, \ldots, x_n$ are the free variables of ϕ

We say that

- ϕ is the **body** of the query,
- $x_1, \ldots, x_n$ are the **output variables**, and
- n is the **arity** of the query.

Example Queries

Let's try to express the following queries as first-order queries:

- Who are the male persons?
- Who are the grandmothers?
- Who are the male persons without children?

What do these queries look like in SQL?

XML Documents as Relational Instances

XML docs are rooted labeled trees, a special case of labeled directed graphs.

Directed graphs (digraphs) consist of

- nodes
- directed edges (described by a binary relation $\mathtt{child}(\cdot,\cdot)$)

Rooted trees are digraphs with

- one source (no incoming edge), called "root"
- an arbitrary number of sinks, called "leaves"
- no cycles.

XML Documents as Relational Instances (cntd)

We assume there is a set of **labels** Σ (labels model the element tags)

A **labeled tree** has exactly one label on each node

The set of all trees with labels from Σ is denoted as T_Σ

We express that a node carries label $a \in \Sigma$ with the unary relation $\mathit{lab}_a(\cdot)$.

We identify all labeled rooted trees t
with instances of the schema with the relations

- $\mathtt{child}(\cdot,\cdot)$
- $\mathit{lab}_a(\cdot)$, $a \in \Sigma$

If t is such a tree, then $\mathit{nod}(t)$ is the set of all nodes of t

We are graciously ignoring strings and other values.
We could model them with unary relations "text'$_{xyz}$'$(\cdot)$"

XPath Queries as First-order Queries

We want to query our movies documents known from the coursework

- Select all movies x: (`//movie`)

- Select all actors x of "Spider Man"

  (`//movie[title/text()='SM']/actor`)

- Select all pairs of actors x appearing in movie y

How many variables do we need to write these queries?

XPath Queries over Trees: Exercises

Write queries asking for

- all movies starring Kirsten Dunst
- all movies starring Kirsten Dunst and William Dafoe
- all movies with Kirsten Dunst, but not William Dafoe

First-order Queries over Trees

Syntax of Tree Formulas

For $a \in \Sigma$ and x, $y \in \text{Vars}$:

$$\phi ::= \mathit{lab}_a(x) \mid \texttt{child}^*(x,y) \mid \texttt{next\_sibling}^*(x,y) \mid \neg\phi \mid \exists x\phi \mid \phi_1 \wedge \phi_2$$

Transitive Closure

We have to add `child`* to the instances if we want to talk about descendants in FO, and similiarly next `next_sibling`* in order to talk about horizontal recursion.

Tree Queries

As before, queries over trees are expressions of the form

$$\{(x_1, \ldots, x_n) \mid \phi\},$$

Core XPath 1.0

Introduced by Gottlob & Koch in [PODS'01, JACM'03]

- "logical core" of XPath 1.0, used to study theoretical properties
- many simplifications
  - removes arithmetic
  - removes functions on data content (e.g., on strings)
  - leaves only the navigational core

Example Queries in Core XPath 1.0

Select all movies

XPath short: `//movie`

XPath long: $\texttt{child}^* :: \texttt{movie}$

FO logic: $\texttt{child}^*(\textit{root}, x) \wedge \textit{lab}_{\texttt{movie}}(x)$

Select all actors acting together with JDepp in a movie

XPath short:
```
//movie[actor/text() = 'JDepp']/
  actor[not[text() = 'JDepp']]
```

XPath long:
```
child*::movie[child::actor[text() = 'JDepp']]/
  child::actor[not[text() = 'JDepp']]
```

FO logic:

$$\exists y_1(\texttt{child}^*(\textit{root}, y_1) \wedge \textit{lab}_{\texttt{movie}}(y_1))$$
$$\wedge\ \exists y_2(\texttt{child}(y_1, y_2) \wedge \textit{lab}_{\texttt{actor}}(y_2) \wedge \texttt{text}_{\texttt{'JDepp'}}(y_2)) \quad \wedge$$
$$\texttt{child}(y_1, x) \wedge \textit{lab}_{\texttt{actor}}(x) \wedge \neg\texttt{text}_{\texttt{'JDepp'}}(x))$$

Core XPath 1.0

Syntax where $a \in \Sigma$

filter $f ::= * \mid a \mid \texttt{not}\,[f] \mid f[p]$

axis $r ::= \texttt{child} \mid \texttt{next\_sibling} \mid r^{-1} \mid r^* \mid \texttt{self} \mid \ldots$

paths $p ::= r :: f \mid p/p' \mid p \cup p' \mid /p$

Semantics for trees $t \in T_\Sigma$

$$\mathit{eval}^t(f) \subseteq \mathit{nod}(t)$$
$$\mathit{eval}^t(r) \subseteq \mathit{nod}(t)^2$$
$$\mathit{eval}^t(p) \subseteq \mathit{nod}(t)^2$$

Core XPath 1.0 Semantics

$$\mathit{eval}^t(*) = \mathit{nod}(t)$$
$$\mathit{eval}^t(a) = \mathit{lab}_a^t \quad \text{(extension of } \mathit{lab}_a \text{ in } t\text{)}$$
$$\mathit{eval}^t(\texttt{not}\ f) = \mathit{nod}(t) \setminus \mathit{eval}^t(f)$$
$$\mathit{eval}^t(f[p]) = \{n \in \mathit{eval}^t(f) \mid \exists n\ .(n, n') \in \mathit{eval}^t(p)\}$$

$$\mathit{eval}^t(\texttt{child}) = \texttt{child}^t \quad \text{(extension of } \texttt{child} \text{ in } t\text{)}$$
$$\mathit{eval}^t(\texttt{next\_sibling}) = \texttt{next\_sibling}^t$$
$$\mathit{eval}^t(r^{-1}) = \mathit{eval}^t(r)^{-1}$$
$$\mathit{eval}^t(r^*) = \mathit{eval}^t(r)^*$$
$$\mathit{eval}^t(\texttt{self}) = \{(n, n) \mid n \in \mathit{nod}(t)\}$$

$$\mathit{eval}^t(r :: f) = \{(n, n') \in \mathit{eval}^t(r) \mid n' \in \mathit{eval}^t(f)\}$$
$$\mathit{eval}^t(p/p') = \mathit{eval}^t(p) \circ \mathit{eval}^t(p') \quad \text{(composition of relations)}$$
$$\mathit{eval}^t(p \cup p') = \mathit{eval}^t(p) \cup \mathit{eval}^t(p')$$
$$\mathit{eval}^t(/p) = \{(\mathit{root}, n) \in \mathit{nod}(t)^2 \mid (\mathit{root}, n) \in \mathit{eval}^t(p)\}$$

Exercises: Expressiveness of Core XPath 1.0

Can one define the following queries in Core XPath 1.0?

- All nodes reachable from the root over a path with labels in a^*b?
- All nodes that are reachable from the root over a path with labels in $(aa)^*$?
- Can one define the same query in Datalog for trees?
- And what about Datalog where all query predicates are unary (monadic datalog)?

Translation to FO Logic

Proposition

Every expression of XPath 1.0 can be translated in linear time to an FO formula with 2 free variables, that define the same binary query.

$$
\begin{aligned}
[\![*]\!]_x &= \textit{true} \\
[\![a]\!]_x &= \textit{lab}_a(x) \\
[\![\texttt{not}\ [f]]\!]_x &= \neg[\![f]\!]_x \\
[\![f[p]]\!]_x &= [\![f]\!]_x \wedge \exists y\, [\![p]\!]_{x,y}
\end{aligned}
$$

$$
\begin{aligned}
[\![\texttt{child}]\!]_{x,y} &= \texttt{child}(x,y) \\
[\![\texttt{next\_sibling}]\!]_{x,y} &= \texttt{next\_sibling}(x,y) \\
[\![r^{-1}]\!]_{x,y} &= r(y,x) \\
[\![r^*]\!]_{x,y} &= r^*(x,y) \\
[\![(r^*)^{-1}]\!]_{x,y} &= r^*(y,x) \\
[\![\texttt{self}]\!]_{x,y} &= (x = y)
\end{aligned}
$$

$$
\begin{aligned}
[\![r :: f]\!]_{x,y} &= [\![r]\!]_{x,y}[\![f]\!]_y \\
[\![p \cup p']\!]_{x,y} &= [\![p]\!]_{x,y} \vee [\![p']\!]_{x,y}
\end{aligned}
$$

$$
\begin{aligned}
[\![p/p']\!]_{x,y} &= \exists z([\![p]\!]_{x,z} \wedge [\![p']\!]_{z,y}) \\
[\![/p']\!]_{x,y} &= (x = \textit{root} \wedge [\![p]\!]_{z,y})
\end{aligned}
$$

Query Answering for Core XPath 1.0

For a start set $S \subseteq \textit{nod}(t)$, let $\textit{eval}^t_S(p) = \{n' \mid n \in S,\ (n, n') \in \textit{eval}^t(p)\}$

Theorem (Gottlob & Koch (TODS'05))

For all expressions p of Core XPath 1.0, trees t, and start sets S, one can compute the monadic query $\textit{eval}^t_S(p)$ in time $O(|t| \cdot |p|)$.

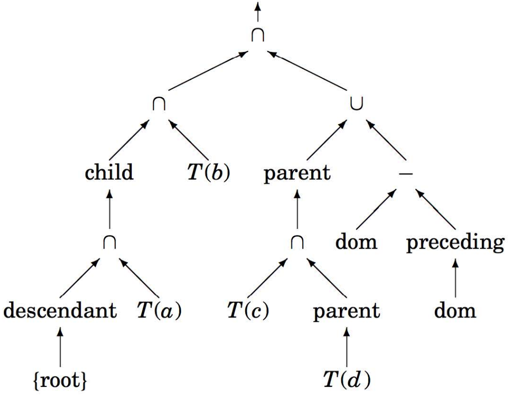
Idea: Uses an algebra where one navigates from S through an operator tree corresonding to p.

Operator Tree of Query [Gottlob & Koch (TODS'05)]

Example 10.3. The Core XPath query

/descendant::a/child::b[child::c/child::d or not(following::*)]

is evaluated as specified by the query tree



Complexity of Full XPath 1.0

Full XPath 1.0 is more difficult because of

- equalities (join on labels)
- data manipulation (string and arithmetic functions)
- context information (context node, position in current node set, size of node set).

Theorem (Gottlob, Koch & Pichler, 2005)

There is an algorithm that evaluates an expression p on a document t with

- time complexity $O(|t|^4 \cdot |p|^2)$
- space complexity $O(|t|^2 \cdot |p|^2)$

sd

Expressivity of Core XPath 1.0 (Marx & de Rijke)

Theorem

For every FO tree query with 2 variables $Q = \{x \mid \phi\}$, there exists a Core XPath expression filter expression f, such that Q and f return the same answers over all rooted labeld trees, and conversely.

In other words, Core XPath 1.0 has the same expressivity as First-Order Logic over trees with 2 variables.

From XPath 1.0 to XPath 2.0

XPath 2.0 extends XPath 1.0 essentially by

- stronger typing (types as in XML schema)
- sequence processing functions (e.g., remove, insert, index-of)
- explicit quantification with variables as in XQuery
  (using `some $x in` p `satisfies, e`, `every $x in` p `satisfies, e`)
- iteration as in XQuery
  (using `for $x in` p `return` e)
- intersection and difference of paths

As for XPath 1.0, one has defined Core XPath 2.0, which adds quantification, iteration, iteration and path intersection and difference to XPath 1.0.

Expressivity of Core XPath 2.0

Theorem (Marx & ten Kaate)

- The evaluation problem for Core XPath 2.0 is PSpace-complete
- Core XPath 2.0 has the same expressivity as full first-order logice over trees.