University of Washington
Department of Computer Science & Engineering

Managing XML and Semistructured Data

Lecture 14: Constraints and Keys

Prof. Dan Suciu

Spring 2001

In this lecture

- Constraints and Keys
  - Path constraints on semistructured data
  - Relative path constraints
  - Proposals for Keys in XML
  - Keys and Schema

Resources

- Keys for XML by Buneman, Davidson, Fan, Hara, Tan, in WWW10, 2001.
- **Data on the Web** Abiteboul, Buneman, Suciu : *section 7.7*

Path Constraints in Semistructured Data

- *Regular Path Queries with Constraints,* Abiteboul and Vianu, PODS'98
- Problem: given a set of *path constraints* optimize regular path expressions
- Especially useful for DAGs, less clear for trees

Path Constraints

- Data instance I = rooted, edge-labeled graph
- Regular path query q = regular expression
- Evaluation: q(I) = a set of nodes

Path Constraints

Path constraints:

- $p = p'$
- $p \subseteq p'$

A data instance I satisfies $p=p'$ if $p(I) = p'(I)$

A data instance I satisfies $p \subseteq p'$ if $p(I) \subseteq p'(I)$

Notation: $I \models p=p'$ or $I \models p \subseteq p'$

Path Constraints

Examples

- $(_)^*.\mathbf{home} = \varepsilon$
  - Says: **home** points back to the root
- $\mathbf{person}.\mathbf{person} \subseteq \mathbf{person}$
  - Says: persons may have other person links, but they only point to other persons
- $\mathbf{person}.(_)^*.(\mathbf{name}.\mathbf{lastname?}) =$ cache46932
  - Says that the path is stored in the cache

Path Constraints

Problem:

- Given a set of path constraints, E:
  - $p_1 =/\subseteq p_1'$
  - …
  - $p_k =/\subseteq p_k'$
- and given queries q, q'
- decide whether E implies $q =/\subseteq q'$
  - Formally: for every I, if $I \models E$, then $I \models q =/\subseteq q'$

Notation: $E \models q =/\subseteq q'$

Path Constraints

Examples

- (_)*.**home** = ε $\models$ q = q'
  where:
  - q = (**home**.**person** | **home**.**company**)*.**address**
  - q' = (**person** | **company**).**address**

  Notice that q' is much simpler !
- **person**.(_)*.(**name**.**lastname?**) = cache46932 $\models$ q = q'
  where:
  - q = **person**.(_)*.(**name**.**lastname?**) .**address**
  - q' = cache46932.**address**

Path Constraints

Solving the implication problem along four dimensions

- The set of constraints E consists of:
  - Word constraints only (i.e. no regular expressions)
  - Arbitrary regular path expressions
- The queries q, q' are:
  - Words only (i.e. no regular path expressions)
  - Arbitrary regular path expressions

Path Constraints

Given E a set of path constraints

- Rewrite system:
  - If $p =/\subseteq p'$ is in E, then p.r → p'.r, for any r
- The rewrite system is *sound* (WHY ??)
- Notice: If $p =/\subseteq p'$ is in E, then r.p → r.p', is not necessarily sound (WHY ???)

Path Constraints

Theorem If E consists of word constraints only, then → is complete

Moreover:

- If q, q' are path expression, can check in PTIME
- Otherwise, can check in PSPACE
- None of this is obvious…

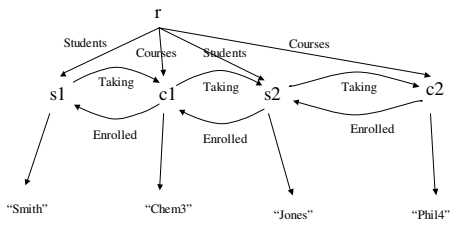
Theorem. In general can check $E \models q = q'$ in EXPSPACE

Relative Path Constraints

- *Path constraints on semistructured and structured data,* Buneman, Fan, Weinstein, PODS'98
- Idea:
  - Path constraints always start from the root
  - Hence very limited
  - Generalize at some arbitrary node

Note: paper uses slightly different notation…

Relative Path Constraints



Relative Path Constraints

ε: Students.Taking $\subseteq$ Courses^{-1}

ε: Courses.Enrolled $\subseteq$ Students^{-1}

Students: Taking $\subseteq$ Enrolled

Courses: Enrolled $\subseteq$ Taking

Definition. Relative path constraint:

$$a: b \subseteq c \quad \text{or} \quad a: b \subseteq c^{-1}$$

$\forall x,y(a(root,x) \wedge b(x,y) \Rightarrow c(x,y))$ or $\forall x,y(a(root,x) \wedge b(x,y) \Rightarrow c(y,x))$

Relative Path Constraints

Implication problem:

- Given a set of relative path constraints E
- Given a path constraint a:b $\subseteq$ c
- Check if E $\models$ a:b $\subseteq$ c

Notice: here we restrict to word problems (are hard enough)

Relative Path Constraints

Bad news:

- The implication problem is, in general, undecidable
- Still: it is decidable in particular cases, such as:
  - When all a's in a:b $\subseteq$ c have the same length
    - This includes the word path constraints, when all a's are equal to ε
  - When all b's have $|b| \leq 1$

Keys in XML Schema

XML:

```
<purchaseReport>
    <regions>
    <zip code="95819">
        <part number="872-AA" quantity="1"/>
        <part number="926-AA" quantity="1"/>
        <part number="833-AA" quantity="1"/>
        <part number="455-BX" quantity="1"/>
    </zip>
    <zip code="63143">
        <part number="455-BX" quantity="4"/>
    </zip>
    </regions>
    <parts>
        <part number="872-AA">Lawnmower</part>
        <part number="926-AA">Baby Monitor</part>
        <part number="833-AA">Lapis Necklace</part>
        <part number="455-BX">Sturdy Shelves</part>
    </parts>
</purchaseReport>
```

XML Schema:

```
<key name="NumKey">
  <selector xpath="parts/part"/>
  <field xpath="@number"/>
</key>
```

Keys in XML Schema

- In general, two flavors:

```
<key name="someDummyNameHere">
  <selector xpath="p"/>
  <field xpath="p1"/>
  <field xpath="p2"/>
  . . .
  <field xpath="pk"/>
</key>
```

```
<unique name="someDummyNameHere">
  <selector xpath="p"/>
  <field xpath="p1"/>
  <field xpath="p2"/>
  . . .
  <field xpath="pk"/>
</key>
```

Note: all Xpath expressions "start" at the element currently being defined
The fields must identify a single node

Keys in XML Schema

- Unique = guarantees uniqueness
- Key = guarantees uniqueness and existence
- All Xpath expressions are "restricted":
  - /a/b | /a/c OK for selector"
  - //a/b/*/c OK for field
  - To "help the implementors" (???)
- Note: better than DTD's ID mechanism

Keys in XML Schema

- Examples

Recall: must have
A single forename,
Single surname

```
<key name="fullName">
      <selector xpath=".//person"/>
      <field xpath="forename"/>
      <field xpath="surname"/>
</key>

<unique name="nearlyID">
      <selector xpath=".//*"/>
      <field xpath="@id"/>
</unique>
```

Foreign Keys in XML Schema

- Examples

```
<keyref name="personRef" refer="fullName">
      <selector xpath=".//personPointer"/>
      <field xpath="@first"/>
      <field xpath="@last"/>
</keyref>
```

Another Proposal for Keys

- *Keys for XML*, Buneman, Davidson, Fan, Hara, Tan, in WWW'10, May, 2001.
- Cleaner definition
- Extends with relative keys
- Addresses satisfiability problem

Another Proposal for Keys

- A **key** is $q \leftarrow \{p_1, \ldots, p_k\}$
- An instance I satisfies the key, if:
  - $\forall\, x_1, x_2 \in q(root)\ ((\exists z_1 \in p_1(x_1).\exists z_2 \in p_1(x_2).\ z_1=z_2) \wedge$
    $\ldots \wedge$
    $(\exists z_1 \in p_k(x_1).\exists z_2 \in p_k(x_2).\ z_1=z_2))$
    $\Rightarrow x_1 = x_2)$

value equality

node equality

Another Proposal for Keys

Examples:

- //person ← {@id}
- //person ← {name} persons w/o name OK
- //person ← {firstname, lastname}
  - What happens with multiple names ?
- //person ← {ε} no distinct persons that have same value
- //person ← {} at most one person
  - What is the difference between these two ?
- //* ← {id}
  - What happens if an id doesn't have an id child ?

it's okay because id elements can have empty id

Another Proposal for Keys

Intuition for $q \leftarrow \{p_1, \ldots, p_k\}$

If I have k values, $z_1, \ldots, z_k$, then there exists at most one $x \in q(root)$ s.t. $z_1 \in p_1(x), \ldots, z_k \in p_k(x)$

Think of retrieving x from $z_1, \ldots, z_k$, using a hash table

Another Proposal for Keys

- Some inference rules for keys
- $q \leftarrow \{p_1, \ldots, p_k\}$ is a key $\Rightarrow q \leftarrow \{p_1, \ldots, p_n\}$ is a key, for $k \leq n$ (superset of key is always a key)
- $q.q' \leftarrow \{p\}$ is a key $\Rightarrow q \leftarrow \{q'.p\}$ is a key (property of trees)

Another Proposal for Keys

Relative key: $q: q' \leftarrow \{p_1, \ldots, p_k\}$

An instance I satisfies the relative key, if $\forall x \in q(I)$, $q' \leftarrow \{p_1, \ldots, p_k\}$ is a key for the instance rooted at x

Another Proposal for Keys

Examples

- /bible/book/chapter: verse $\leftarrow$ {number}
- /bible/book: chapter $\leftarrow$ {number}
- /bible: book $\leftarrow$ {name}

Another Proposal for Keys

- No relative keys in XML-Schema
- But could work around:

```
<key name="dummyName">
      <selector xpath="/bible/book/chapter"/>
      <field xpath="number"/>
      <field xpath="../number"/>
      <field xpath="../../name"/>
</key>
```

Combining Keys and Schemas

- *On XML Integrity Constraints in the Presence of DTDs,* Fan and Libkin, PODS'2001
- Keys + DTDs sometimes imply unexpected facts
- Main story: implication is undecidable

Combining Keys and Schemas

```
<teachers>
  <teacher name="Joe">  <subject expert="Jim"> DB </subject>
                        <subject expert="Karl"> Graphics </subject>
  </teacher>
  <teacher name="Jim">  <subject expert="Joe"> AI </subject>
                        <subject expert="Fred"> OS </subject>
  </teacher>
  . . . .
</teachers>
```

```
<!ELEMENT teachers (teacher+)>
<!ELEMENT teacher (subject,subject)>
```

Combining Keys and Schemas

Keys and foreign keys:

- Keys:
  - //teacher ← @name
  - //subject ← @expert
- Foreign keys:
  - //@expert ⊆ //teacher/@name

- But this is impossible !
- In general: undecidable to check if it is possible