Reasoning about XML Constraints

Sebastian Link

School of Information Management
Centre for Logic, Language and Computation
Victoria University of Wellington
New Zealand

(joint work with Sven Hartmann)

School of Informatics
Clausthal University of Technology
Germany

This research is supported by the Marsden Fund Council from Government funding, administered by the Royal Society of New Zealand.

Tokyo, 22 June 2009
Motivation

- XML: de-facto standard for Web data exchange and integration
- high degree of syntactic flexibility, low degree of semantic capabilities
- challenge for computer scientists: provide full-fledged tools that can store, manage and process XML data in its native format


- integrity constraints enhance semantic capabilities:
  - find natural classes of constraints that can be maintained efficiently
  - balance trade-off between expressiveness and efficiency
An XML fragment

```xml
<db>
  <year calendar="2007">
    <semester no="1">
      <course name="maths">
        <teacher> <title>Principal</title> <name>Skinner</name> </teacher>
        <student sid="007" first="Bart" last="Simpson" grade="A+" />
      </course>
    </semester>
    <course name="physics">
        <teacher> <title>Principal</title> <name>Skinner</name> </teacher>
        <student sid="007" first="Bart" last="Simpson" grade="A-" />
    </course>
  </semester>
  <semester no="2">
    <course name="PE">
        <teacher> <title>Principal</title> <name>Skinner</name> </teacher>
        <student sid="007" first="Bart" last="Simpson" grade="A+" />
        <student sid="247" first="Lisa" last="Simpson" grade="B-" />
    </course>
  </semester>
</db>
```
Its corresponding XML Tree
Some Constraints

- relatively to each year: a semester is determined by its number
- each year node has either no semester nodes or precisely two
- relatively to each course: a student is determined by its sid
- each student deciding to study in a semester enrolls into at least two and at most 4 courses in that semester
- no student enrolls more than three times into the same course
- no course is offered more than once a year
Scenario 1: Data Cleaning


→ dirty date costs (US) businesses billions of dollars annually
→ data cleaning contributes to 30-80 percent of development time in data warehouse projects

▸ data cleaning tools for removing inconsistencies and errors from data

▸ provide means that prevent dirty data from entering the database

▸ *constraints* restrict databases to those considered meaningful
  → above business rules capture properties of XML data
  → only allow those XML documents conforming to all business rules
Scenario 2: Predicting Query Costs

- Bart Simpson wishes to query XML source for results in 2007 using his cell-phone but only when costs reasonable. Service provider: only retrieve data if service paid for.

- Consider the following XQuery query:

  ```
  for $s in doc("enrol.xml")/year[@calendar="2007"]/course/student
  where $s/sid="007"
  return ⟨grade⟩{$s/grade}⟨/grade⟩
  ```

- XML DBMSs capable of reasoning about cardinality constraints foresees maximal number of eight answers (without processing any data!)

- Approximate costs returned to Bart Simpson who decides accordingly.

- Service provider minimizes costs for unpaid services.

Tokyo, 22 June 2009
Scenario 3: Query Optimisation

► teachers teach at most 3 courses per year
► students take up to 4 courses per semester, with two semester per year
► XML query optimiser should transform the XQuery query

```xml
for $c in doc("enrol.xml")/year[@calendar="2007"]/semester/course
where $c/student/sid="247" and $c/teacher="Principal Skinner"
return ⟨course⟩{$c/name}⟨/course⟩
```

► into

```xml
for $c in doc("enrol.xml")/year[@calendar="2007"]/semester/course
where $c/teacher="Principal Skinner" and $c/student/sid="247"
return ⟨course⟩{$c/name}⟨/course⟩
```

► based on smaller selectivity
Scenario 4: Predicting the No of Updates

- Databases often subject to updates

- Following XQuery updates last name of Lisa Simpson to Milhouse in all semester 2 courses of 2007:

```xml
for $s in doc("enrol.xml")/year[@calendar="2007"]/semester[@no="2"]/student[@sid="247"]
return do replace value of $s/last with "Milhouse"
```

- Maximal number of four updates (decide implication efficiently)
Scenario 5: An encrypted XML fragment

```xml
<db>
  <year calendar="2007">
    <semester no="1">
      <course name="maths">
        <teacher>
          <title>Principal</title>
          <name>Skinner</name>
        </teacher>
        <student sid="007">
          <first>Bart</first>
          <last>Simpson</last>
        </student>
        <grade>
          <xenc:EncryptedData>
            <xenc:CipherData>
              <xenc:CipherValue>AbC234ndZ…</xenc:CipherValue>
            </xenc:CipherData>
          </xenc:EncryptedData>
        </grade>
      </course>
    </semester>
  </year>
</db>
```
Scenario 5: Predicting the No of Encryptions

- Web data often encrypted (grades of students)

- following XQuery query retrieves grades of Bart Simpson in 2007:

  ```
  for $s in doc("enrol.xml")/year[@calendar="2007"]//course/student
  where $s/sid="007"
  return ⟨grade⟩{$s/grade}⟨/grade⟩
  ```

- efficient reasoning about constraints enables DBMS to infer that at most eight decryptions necessary
Scenario 6: XML Transformations

- year-nodes subsume between two and eight course-nodes containing student/sid-subnodes with same value

- querying original XML tree for course info based on specific year and specific sid unnatural

- sid-updates difficult

- create XML view
Scenario 6 continued: Query Rewriting

- rewrite

```xml
for $s$ in doc(“enrol.xml”)/year[@calendar=”2007”]/course/student
where $s/sid=”007”
return ⟨grade⟩{$s/grade}⟨/grade⟩
```

- as

```xml
for $c$ in doc(“view.xml”)/year[@calendar=”2007”]/student[@sid=”007”]/course
return ⟨grade⟩{$c/grade}⟨/grade⟩
```

- early selection of student-elements based on their sid
- even more efficient in case @sid-values are encrypted
- easy updates of sid-values
Scenario 7: Consistent Query Answering

- approach to querying inconsistent databases without repairing them
- retrieves only certain answers, those present in all repairs

**Bart, Lisa, Maggie:** //course[@name = 'physics']/student/first

**however:**

there mustn’t be different student-nodes with same @sid in any course
CQA: One Solution

repair may refer to removal of any offending student-node:

CQA returns only Maggie as a certain answer:

```
//course[@name = 'physics']/student/first
```
CQA: A different Policy

- repair may refer to replacement of any offending \@$sid@$-value by a new \@$sid@$-value not present in the tree; infinitely many repairs, e.g.:

\[
\text{db} \rightarrow \text{course} \\
\text{course}[\text{name} = \text{"physics"}] \\
\text{teacher}[\text{name} = \text{"Principal"}, \text{title} = \text{"Skinner"}] \\
\text{student}[\text{firstlast} = \text{"Bart"}, \text{grade} = \text{"A"}, \text{id} = \text{"007"}] \\
\text{student}[\text{firstlast} = \text{"Lisa"}, \text{grade} = \text{"B"}, \text{id} = \text{"247"}] \\
\text{student}[\text{firstlast} = \text{"Maggie"}, \text{grade} = \text{"A+"}, \text{id} = \text{"107"}]
\]

- Bart, Lisa and Maggie present in all of them:

```sql
//course[@name = 'physics']/student/first
```

Tokyo, 22 June 2009
Practical Benefits of Cardinality Constraints

- represent characteristics that XML data naturally exhibits
- restrict permitted XML database to those considered meaningful
- advance consistency of XML databases
- identify dirty data
- can be specified independently from schema specifications
- enable logical query optimization
- enable physical query optimization
- enable us to infer bounds on number of query answers (updates, en/decryptions) without any data processing
- unlock “good” XML database design, i.e., absence of data redundancies, processing difficulties and inefficient query processing
- new approach to dealing with inconsistent data (CQA)
**Fundamental Challenges and Outline**

- Introduce cardinality constraints based on XML tree model
  → Core CS problem: *expressiveness vs. userfriendliness*

- Investigate decision problems to unlock XML applications
  - Satisfiability problem and finite implication problem

- Identify sources of intractability: coNP-hardness results
  → Core CS problem: *expressiveness vs. tractability*

- Identify large tractable subclass
  - Finite axiomatisation
    → Core CS problem: *syntax vs. semantics*
  - Characterise implication in terms of shortest path
  - Devise algorithm to efficiently decide implication
    → Core CS problem: *problems vs. algorithmic solutions*
XML Trees

- XML documents can be illustrated as rooted trees
  - an XML tree is a 6-tuple $T = (V, \text{lab}, \text{ele}, \text{att}, \text{val}, r)$
  - connected digraphs, no cycles, unique path from $r$ to each node
- we distinguish attribute nodes, element nodes, and text nodes
  - attribute nodes $v$ have attribute name $\text{lab}(v) \in A$
  - element nodes $v$ have element name $\text{lab}(v) \in E$
  - text nodes $v$ have fixed name $\text{lab}(v) = S$, root $r$ is element node
- element nodes $v$ have children
  - a set $\text{att}(v) \subseteq V$ of attribute nodes
  - a list $\text{ele}(v) \subseteq V$ of element and text nodes
  - each child $w$ induces an edge $(v, w)$ in the XML tree
- attribute and text nodes have no children (they are leaves), but each attribute or text node $v$ carries a value $\text{val}(v) \in \text{String}$
An Example of an XML Tree
**Value Equality**

- each node \( v \) is the root of a unique (maximal) sub-tree \( T(v) \) of \( T \)
- two nodes \( v_1 \) and \( v_2 \) are *value-equal* iff their induced sub-trees \( T(v_1) \) and \( T(v_2) \) are isomorphic
  - we denote this by \( v_1 =_v v_2 \)
- roughly speaking, we have:
  - \( \text{lab}(v_1) = \text{lab}(v_2) \)
  - if \( v_1, v_2 \) are attribute or text nodes, then \( \text{val}(v_1) = \text{val}(v_2) \)
  - if \( v_1, v_2 \) are element nodes, then they possess
    - sets of value-equal attribute children
    - lists of value-equal element and text children
- we define the *value-intersection* of two node sets \( V_1 \) and \( V_2 \)
  - \( V_1 \cap_v V_2 = \{(v_1, v_2) \mid v_1 \in V_1, v_2 \in V_2, v_1 =_v v_2\} \)
Example: Value-Equality

- student-nodes: \( v_1 = v_3, v_2 \neq v_1 \text{ and } v_2 \neq v_3 \)
- no two distinct course-nodes are value-equal
- for \( V_1 = \{v_1, v_2\} \) and \( V_2 = \{v_3, v_4\} \) we have \( V_1 \cap V_2 = \{(v_1, v_3)\} \)
Path languages for selecting nodes

- we need a suitable path language to select nodes in the XML tree
  - expressive enough to allow reasonable navigation
  - simple enough to allow efficient reasoning
- we use the path languages $PL$ and $PL_s$ here

<table>
<thead>
<tr>
<th>Path Language</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PL_s$</td>
<td>$P ::= \epsilon \mid P.\ell$</td>
</tr>
<tr>
<td>$PL$</td>
<td>$Q ::= \epsilon \mid Q.\ell \mid Q._*$</td>
</tr>
</tbody>
</table>

- $\ell \in E \cup A \cup \{S\}$ denotes any node name
- $*_*$ denotes a wildcard, $\epsilon$ denotes the empty path

- path expressions in $PL_s$ are simple path expressions
- each expression $Q$ in $PL$ represents set of simple path expressions
  - wildcards $*_*$ in $Q$ replacable by arbitrary simple path expressions
  - write $P \in Q$ for each simple $P$ that can be generated from $Q$
Example: Path Expressions

- simple path expressions:
  \[ \rightarrow \text{year.semester} \]
  \[ \rightarrow \text{semester.student} \]
  \[ \rightarrow \text{year.semester.course.student.grade} \]

- path expressions:
  \[ \rightarrow _\ast .\text{year.semester} \]
  \[ \rightarrow _\ast .\text{course._\ast .grade} \]

- simple path expressions in the regular language of path expressions:
  \[ \rightarrow \text{year.semester} \in _\ast .\text{year.semester} \]
  \[ \rightarrow \text{year.semester.course.student.grade} \in _\ast .\text{course._\ast .grade} \]
Selecting nodes in an XML tree

- path expression $Q$ said to be \textit{valid} iff no attribute name and no $S$ occur in position other than terminal
  - attribute and text nodes are always leaves of the XML tree

- $u[Q]$: those nodes in tree $T$ \textit{reachable} from node $u$ via simple $P \in Q$

- $Q_1 \subseteq Q_2$ iff $u[Q_1] \subseteq u[Q_2]$ for all trees $T$ and all nodes $u$ of $T$

- Path containment for $PL$:
  decide for arbitrary $Q_1, Q_2 \in PL$ whether $Q_1 \subseteq Q_2$ holds

- Path containment for $PL$ can be decided in quadratic time
Example: Navigation between Nodes

\[ u[\_\_\_\_.course.\_\_\_.first] = \{v_1, v_2, v_3, v_4\} \]
\[ \text{semester.course.}\_\_\_.first \subseteq \_\_\_.course.\_\_\_.first \]
\[ \text{not } \_\_\_.course.\_\_\_.first \subseteq \text{semester.course.}\_\_\_.first \]
The Path Containment Problem

- some results on the complexity of the path containment problem:
  - \textbf{PTIME} \(XP(//,//), XP(//,//,[]), XP(/,[],*), XP//(,//,[])\)
  \(XP//(,//,*), DTD)\)
  - \textbf{coNP-complete} \(XP(/,//,[],*), XP(/,//,[],*), XP(/,[],DTD)\)
  - \textbf{coNP-hard} \(XP(/,[]), XP//(,[]), XP//(,[]), DTD)\)
  - \textbf{EXPTIME-complete} \(XP(/,//,[],*,|), DTD)\)
  - \textbf{EXPTIME-hard} \(XP(/,//,[],*,|), DTD)\)

Cardinality constraints for XML

- A cardinality constraint $\varphi$ has the form

$$\text{card}(Q, (Q', \{Q_1, \ldots, Q_k\})) = (\min, \max)$$

- $Q \in PL$ is the context path
- $Q' \in PL$ is the target path
- $Q_i \in PL$ is a key path (for $i = 1, \ldots, k$)
- $Q.Q'$ valid ($k = 0$) or $Q.Q'.Q_i$ valid path (for $i = 1, \ldots, k$)

- Two nodes $v_1, v_2$ are $\{Q_1, \ldots, Q_k\}$-confusable iff

$$v_1[Q_i] \cap v_2[Q_i] \neq \emptyset \text{ for all } i = 1, \ldots, k$$

- The XML tree $T$ satisfies $\varphi$ iff for all nodes $q \in r[Q]$ and all nodes $q' \in q[Q']$ there are no less than $\min$ and no more than $\max$ nodes in $q[Q']$ that are $\{Q_1, \ldots, Q_k\}$-confusable with $q'$
Example 1

in every semester every student who studies must enrol in 2 to 4 courses

$T$ does not satisfy $\text{card}(\_*.\text{semester}, (\text{course}, \{\_*.\text{sid}\}))=(2,4)$
Example 2

- $T$ satisfies $\text{card}(\text{year},(_\ast \cdot \text{course}, \{\text{teacher}\})) = (3,6)$
- $T$ violates $\text{card}(\_\ast \cdot \text{semester},(\text{course}, \{\text{teacher}\})) = (2,2)$
Example 3

▶ XML keys covered by cardinality constraints: \( \text{min} = \text{max} = 1 \)


→ S. Hartmann, S. Link. Efficient Reasoning about a Robust XML Key Fragment. TODS 34(2:10), 2009

▶ \( T \) satisfies \( \text{card}(\_\star .\text{course},(\text{student},\{\text{sid}\}))=(1,1) \)
Towards a Visual Specification Language

- allow users to specify their requirements graphically
- allow designers to present her specifications to users visually
- teachers that teach in a semester, teach between 2 and 4 different courses in that semester
- formally: \[ \text{card}(\_\_\_.\text{semester},(\text{course},\{\_\_\_.\text{title},\text{teacher.\text{name}}\}))=(2,4) \]
- visually as a pattern:

```
  \_\_\_.\text{semester}  \_\_\_.\text{course}
      \_\_\_.\text{teacher}  \_\_\_.\text{title}
          \_\_\_.\text{name}
```

Tokyo, 22 June 2009
Decision Problems

▸ (finite) satisfiability problem:
   \( \rightarrow \) Given any finite \( \Sigma \), is there (finite) \( T \) satisfying all \( \sigma \in \Sigma \)?

▸ \( \Sigma \) (finitely) implies \( \varphi \in \Sigma^* \) (semantic closure):
   \( \rightarrow \) each (finite) XML tree that satisfies \( \Sigma \) also satisfies \( \varphi \)

▸ (finite) implication problem:
   \( \rightarrow \) Given any finite \( \Sigma \cup \{ \varphi \} \): does \( \Sigma \) (finitely) imply \( \varphi \)?

▸ efficient solution to decision problems unlocks XML applications

▸ let \( \Sigma^+_R \) be the syntactic closure of \( \Sigma \)
  - with respect to a fixed system \( R \) of inference rules
  - \( \varphi \in \Sigma^+_R \) can be inferred from \( \Sigma \) using the rules in \( R \)

▸ \( R \) is sound iff \( \Sigma^+_R \subseteq \Sigma^* \), and complete iff \( \Sigma^* \subseteq \Sigma^+_R \)
A Computer Scientist’s Pilgrimage

Mark Musa, in his translation of Dante’s *Divine Comedy*:

Dante the Pilgrim, in order to arrive at the Divine Light, will come to an understanding of sin on his journey through Hell and will see the penance imposed on repentant sinners on the Mount of Purgatory.

- our journey through Hell (it’s not that bad!):
  - our sin: too much expressiveness
  - understanding: identifying intractabilities

- our Mount Purgatory:
  - restriction to tractable cases
  - do penance by studying properties

- our “Divine Light”:
  - finite axiomatization
  - efficient algorithms
Sources of Intractability: Lower and Upper Bounds

\[ C_1 = \{ \text{card}(\epsilon, (P', \{P_1, \ldots, P_k\})) = (\min, \max) \mid k \geq 1 \} \]

Theorem:
The finite implication problem for the class \( C_1 \) of all simple absolute cardinality constraints with a non-empty set of key paths is co\( NP \)-hard.

suggests that computational intractability may result from the specification of both lower and upper bounds
Sources of Intractability: Empty sets of key paths

\[ C_2 = \{ \text{card}(\epsilon, (P', \{P_1, \ldots, P_k\})) = (1, \text{max}) \mid k \geq 0 \} \]

Theorem:
The finite implication problem for the class \( C_2 \) of all simple absolute cardinality constraints where the lower bound is fixed to 1 is \( coNP \)-hard.

suggests that computational intractability may result from the permission of empty sets of key paths
Sources of Intractability: General key path expressions

\[ C_3 = \{ \text{card}(\epsilon, (Q', \{Q_1, \ldots, Q_k\})) = (1, \max) \mid k \geq 1 \} \]

**Theorem:**

The finite implication problem for the class $C_3$ of all absolute cardinality constraints that have a non-empty set of key paths and where the lower bound is fixed to 1 is coNP-hard.

suggests that computational intractability may result from permission to have arbitrary path expressions in both target- and key paths.
Numerical keys for XML

- A numerical key \( \varphi \) has the form
  \[
  \text{card}(Q, (Q', \{ P_1, \ldots, P_k \})) \leq \text{max}
  \]
  - \( Q \in PL \) is the context path
  - \( Q' \in PL \) is the target path
  - \( P_i \in PL \) is a key path (for \( i = 1, \ldots, k \) and \( k \geq 1 \))
  - \( Q.Q'.P_i \) valid path (for \( i = 1, \ldots, k \))

- XML tree \( T \) satisfies \( \text{card}(Q, (Q', \{ P_1, \ldots, P_k \})) \leq \text{max} \) iff \( T \) satisfies \( \text{card}(Q, (Q', \{ P_1, \ldots, P_k \})) = (1, \text{max}) \)

- Theorem:
  Every finite set of numerical keys is finitely satisfiable.

- Theorem:
  Implication and finite implication coincide for numerical keys.
A finite axiomatisation for numerical keys

<table>
<thead>
<tr>
<th>Card</th>
<th>Condition</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{card}(Q, (Q', S)) \leq \infty)</td>
<td>(infinity)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', S')) \leq \max)</td>
<td>(superkey)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', S \cup {P})) \leq \max)</td>
<td>(context-path-containment)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', S \cup {\epsilon, P})) \leq \max)</td>
<td>(prefix-epsilon)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (\epsilon, S')) \leq 1)</td>
<td>(epsilon)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.P, {P'})) \leq \max)</td>
<td>(subnodes)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', {P.P'})) \leq \max)</td>
<td>(subnodes-epsilon)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.P, {\epsilon, P'})) \leq \max)</td>
<td>(target-path-containment)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', S)) \leq \max)</td>
<td>(weakening)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.Q'', S)) \leq \max)</td>
<td>(context target)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', S)) \leq \max)</td>
<td>(Q'' \subseteq Q')</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q', {\epsilon, P.P'})) \leq \max)</td>
<td>(multiplication)</td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.P, {P_1, \ldots, P_k})) \leq \max)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.P, {P_1, \ldots, P_k})) \leq \max')</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{card}(Q, (Q'.P, {P_1, \ldots, P_k})) \leq \max \cdot \max')</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

- sports league competition:
- per season same home and away team both play in at most 3 months:
  \[ \sigma_1 = \text{card}(\_*.\text{season}, (\text{month}, \{\text{match.home,match.away}\})) \leq 3 \]
- per month same home team plays same away team at most twice:
  \[ \sigma_2 = \text{card}(\_*.\text{season.month}, (\text{match}, \{\text{home,away}\})) \leq 2 \]
- same home team plays same away team at most 5 times per season:
  \[ \phi = \text{card}(\_*.\text{season}, (\text{month.match}, \{\text{home,away}\})) \leq 5 \]
- \( \phi \) cannot be inferred from \( \Sigma = \{\sigma_1, \sigma_2\} \)
  \[ \leadsto \text{How can we show that } \phi \text{ is not implied by } \Sigma ? \]
  \[ \leadsto \text{we construct a counter-example} \]
Shortest Paths vs Derivability

we construct a cardinality graph $G_{\Sigma,\varphi}$ from $\Sigma$ and $\varphi$

if $d(q_\varphi, q'_\varphi) \leq \max_\varphi$ in $G_{\Sigma,\varphi}$, then $\varphi \in \Sigma^+$

in other words:

- if $\varphi$ cannot be inferred from $\Sigma$, then
  $d(q_\varphi, q'_\varphi) \geq \max_\varphi + 1$ in $G_{\Sigma,\varphi}$

Example:

- $\varphi = \text{card}(\_*.\text{season}, (\text{month}\_.\text{match}, \{\text{home,away}\})) \leq 5$
  not derivable from
- $\sigma_1 = \text{card}(\_*.\text{season}, (\text{month}, \{\text{match.home,match.away}\})) \leq 3$
- $\sigma_2 = \text{card}(\_*.\text{season.month}, (\text{match}, \{\text{home,away}\})) \leq 2$
Completeness

- $T$ satisfies
  - $\sigma_1 = \text{card}(\_\star.\text{season}, (\text{month}, \{\text{match.home,match.away}\}))) \leq 3$
  - $\sigma_2 = \text{card}(\_\star.\text{season.month}, (\text{match}, \{\text{home,away}\}))) \leq 2$

- $T$ violates
  - $\varphi = \text{card}(\_\star.\text{season}, (\text{month.match}, \{\text{home,away}\}))) \leq 5$
An Algorithm for the Implication Problem

- strong method: $\varphi \in \Sigma^*$ iff $d(q_\varphi, q'_\varphi) \leq \max_\varphi$ in $G_{\Sigma, \varphi}$
- this is how the algorithm works:
  - **Input:** $\Sigma$, $\varphi$
  - **Output:** YES if $\varphi \in \Sigma^*$; No otherwise
  - **Method:**
    1. construct the cardinality graph $G_{\Sigma, \varphi}$
    2. if $d(q_\varphi, q'_\varphi) \leq \max_\varphi$ then return YES else return NO
- shortest path: apply Dijkstra’s algorithm using start node $q_\varphi$ ($O(|\varphi|^2)$)
- cardinality graph constructable in time $O(||\Sigma|| \cdot |\varphi|^3)$
  - we use the evaluation algorithm for Core XPath queries
- in fact: cardinality graph constructable in time $O(||\Sigma|| \cdot |\varphi|)$
A finite axiomatisation for XML keys

- \( \text{key} \models_T (Q, (Q', \{P_1, \ldots, P_k\})) \) iff \( \models_T \text{card}(Q, (Q', \{P_1, \ldots, P_k\})) \leq 1 \)

\[
\begin{align*}
(Q, (\epsilon, S')) & \quad (Q, (Q', S \cup \{\epsilon, P\})) & \quad (Q, (Q', S)) \\
\text{epsilon} & \quad \text{(prefix-epsilon)} & \quad \text{(superkey)} \\
(Q, (Q', P, \{P'\})) & \quad (Q, (Q', S)) & \quad (Q, (Q', S')) \\
(Q, (Q', \{P.P'\})) & \quad (Q'' \subseteq Q) & \quad (Q'', (Q', S)) & \quad (Q'', \subseteq Q') \\
\text{subnodes} & \quad \text{(context-path-containment)} & \quad \text{(target-path-containment)} \\
(Q, (Q'.Q'', S')) & \quad (Q, (Q'.P, \{\epsilon, P'\})) & \quad (Q, (Q', \{P.P_1, \ldots, P.P_k\})), \quad (Q, (Q', \{P_1, \ldots, P_k\})), \\
(Q.Q', (Q'', S')) & \quad (Q, (Q', \{\epsilon, P.P'\})) & \quad (Q, (Q'.P, \{P_1, \ldots, P_k\})), & \quad (Q, (Q'.P, \{P_1, \ldots, P_k\})) \\
\text{(context target)} & \quad \text{(subnodes-epsilon)} & \quad \text{(interaction)}
\end{align*}
\]
Reachability instead of Shortest Paths

- all witness edges have weight 1,
- discard with labels to obtain reachability graph
- this is how the algorithm works:
  - **Input:** $\Sigma$, $\varphi$
  - **Output:** YES if $\varphi \in \Sigma^*$; No otherwise
  - **Method:**
    1. construct the reachability graph $G_{\Sigma,\varphi}$
    2. if $q'_{\varphi}$ reachable from $q_{\varphi}$ then return(YES) else return(NO)
- reachability test: depth-first search of $G_{\Sigma,\varphi}$ with root $q_{\varphi}$ in $\mathcal{O}(|\varphi|^2)$
- reachability graph constructable in time $\mathcal{O}(|\Sigma| \cdot |\varphi|)$
- we use the evaluation algorithm for Core XPath queries
Structural XML Keys

- keys: target nodes uniquely identified by values reachable via key paths
- structural keys identify target nodes independently of any data values
- \((\ast . \text{course}, (\text{name}, \emptyset))\), but not \((\ast . \text{semester}, (\ast . \text{name}, \emptyset))\)

- allow structural keys: XML key is expression \((Q, (Q', S))\) where
  - \(Q, Q'\) are PL expressions,
  - \(S\) is a finite set of PL expressions such that
  - \(S \neq \emptyset\): \(Q.Q'.P\) are valid PL expressions for all \(P \in S\) or
  - \(S = \emptyset\): \(Q.Q'\) is a valid PL expression.

- satisfiability of XML keys defined as before
### Axiomatising XML keys and Structural keys

<table>
<thead>
<tr>
<th>(Q, (ε, S))</th>
<th>(Q, (Q', S ∪ {P, P.P'}))</th>
<th>(Q, (Q', S)</th>
<th>(Q'' \subseteq Q)</th>
<th>Q'' (\subseteq Q')</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q, (Q', P, {P'}))</td>
<td>(Q, (Q', {ε, P'}))</td>
<td>(Q, (Q', {ε, P.P'}))</td>
<td>(Q, (Q'', {Q'' \subseteq Q})</td>
<td>(Q, (Q'', {Q'' \subseteq Q'})</td>
</tr>
<tr>
<td>(Q, (Q'', S))</td>
<td>(Q, (Q.Q', S))</td>
<td>(Q, (Q.Q', P, S))</td>
<td>(Q, (Q.Q', P, S))</td>
<td>(Q, (Q.Q', (P, S)))</td>
</tr>
<tr>
<td>(Q, (Q.Q', Q''))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
</tr>
<tr>
<td>(Q, (Q.Q', \emptyset))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
</tr>
<tr>
<td>(Q, (Q.Q', S))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
<td>(Q, (Q.Q', (Q'' \subseteq Q, S)))</td>
</tr>
</tbody>
</table>

### Notes
- **(epsilon)**: prefix
- **(superkey)**
- **(subnodes)**
- **(context-path-containment)**
- **(interaction)**
- **(border)**
- **(multiple subnodes)**
- **(neighbour)**

Tokyo, 22 June 2009
Deciding Implication

- method may fail: reachability graph can violate structural key
- pre-processing: merge distinct key paths to satisfy structural keys

**Input:** $\Sigma$, $\phi$

**Output:** YES if $\phi \in \Sigma^*$; No otherwise

**Method:**

1. Construct $T_{\Sigma,\phi}$ from $\Sigma$ and $\phi$;
2. **IF** $\exists (Q_\sigma, (Q'_\sigma, \emptyset)) \in \Sigma. \exists w_\sigma \in [Q_\sigma]$ in $T_{\Sigma,\phi}$ and $\exists w'_\sigma, w''_\sigma \in w_\sigma[Q'_\sigma]$ in $T_{\Sigma,\phi}$ with $w'_\sigma \neq w''_\sigma$
3. **THEN RETURN**(yes)
4. **ELSE** Construct $G_{\Sigma,\phi}$ from $\Sigma$ and $\phi$;
5. **IF** $q_\phi$ is reachable from $q'_\phi$ in $G_{\Sigma,\phi}$ **THEN RETURN**(yes)
6. **ELSE RETURN**(no).

- implication still decidable in time $O(|\phi| \times (||\Sigma|| + |\phi|))$
Findings for Cardinality Constraints

- Form a natural class of XML constraints that generalise XML keys.
- Useful for many XML applications:
  - XQuery, XPath, XML Encryption, XQuery update facility
  - Cost estimation, query optimisation, query rewriting
- Reasoning about cardinality constraints intractable in general:
  - Specification of both lower and upper bounds
  - Empty set of key paths
  - General key path expressions
- Identified numerical keys as a large tractable subclass:
  - Always satisfiable
  - Implication and finite implication coincide
  - Established finite axiomatisation
  - Characterised implication in terms of shortest paths
  - Efficient solutions to decision problem unlocks applications
Future Work?
Future Work

- push expressiveness while maintaining tractability
- develop visual specification languages for cardinality constraints
- validity: is an XML document valid wrt a CC (tree automata)
- discover all CCs satisfied by a given XML document (data mining)
- interaction with schema specifications such as DTDs and XSDs
- interaction with other types of constraints
- automatic support for query optimization
- physical database tuning: indexes for efficient query performance
- investigate consistent query answering
- database design: transform poorly-designed databases into well-designed ones that avoid data redundandancies, processing difficulties and inefficient query processing
Questions?