

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.

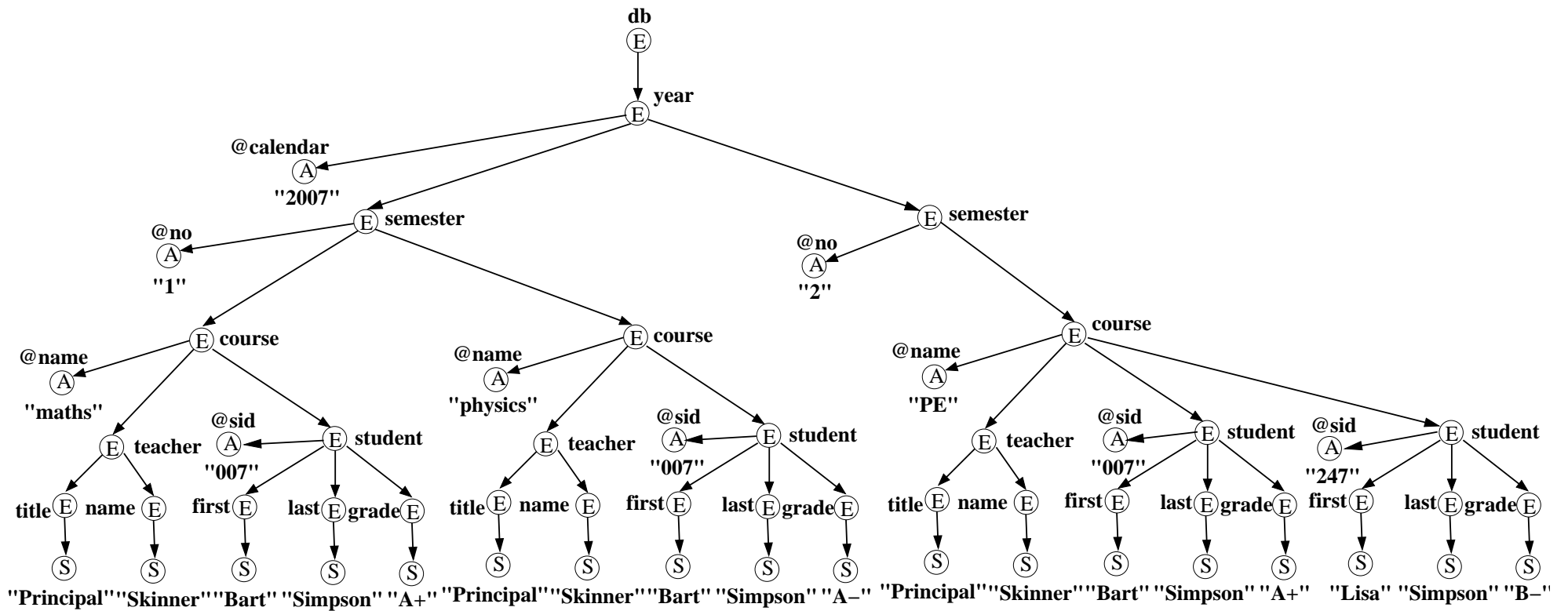
Motivation

- 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
 - S. Abiteboul, P. Buneman, D. Suciu. *Data on the Web: From relations to semistructured data and XML*. Morgan Kaufmann, 1999.
 - S. Ceri, P. Fraternali, S. Paraboschi. *XML: Current developments and future challenges for the database community*. EDBT, 3-17, 2000.
 - W. Fan. *XML constraints*. DEXA Workshops, 805-809, 2005.
 - D. Suciu. *On database theory and XML*. SIGMOD Record, 30(3):39-45, 2001.
 - V. Vianu. *A web odyssey: from Codd to XML*. PODS, 1-15, 2001.
 - J. Widom. *Data management for XML: Research direction*. Data Engineering Bulletin, 22(3):44-52, 1999.
- 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

```
<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> <grade>A+</grade> </student>
      </course>
      <course name=physics>
        <teacher> <title>Principal</title> <name>Skinner</name> </teacher>
        <student sid=007> <first>Bart</first> <last>Simpson</last> <grade>A-</grade> </student>
      </course>
    </semester>
    <semester no=2>
      <course name=PE>
        <teacher> <title>Principal</title> <name>Skinner</name> </teacher>
        <student sid=007> <first>Bart</first> <last>Simpson</last> <grade>A+</grade> </student>
        <student sid=247> <first>Lisa</first> <last>Simpson</last> <grade>B-</grade> </student>
      </course>
    </semester>
  </year>
</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

- ▶ W.W. Eckerson: *Data Quality and the Bottom Line: Achieving Business Success through a Commitment to High Quality Data*. Technical Report. The Data Warehousing Institute. 2002.
<http://www.tdwi.org/research/display.aspx?ID=6064>
 - ↳ dirty data costs (US) businesses billions of dollars annually
 - ↳ data cleaning contributes to 30-80 percent of development time in datawarehouse 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

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

```
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

```
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:

```
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

```
<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>
        <grade>
          <xenc:EncryptedData>
            <xenc:CipherData>
              <xenc:CipherValue> AbC234ndZ... </xenc:CipherValue>
            </xenc:CipherData>
          </xenc:EncryptedData>
        </grade>
      </student>
    </course>
  </semester>
</year>
</db>
```

Scenario 5: Predicting the No of Encryptions

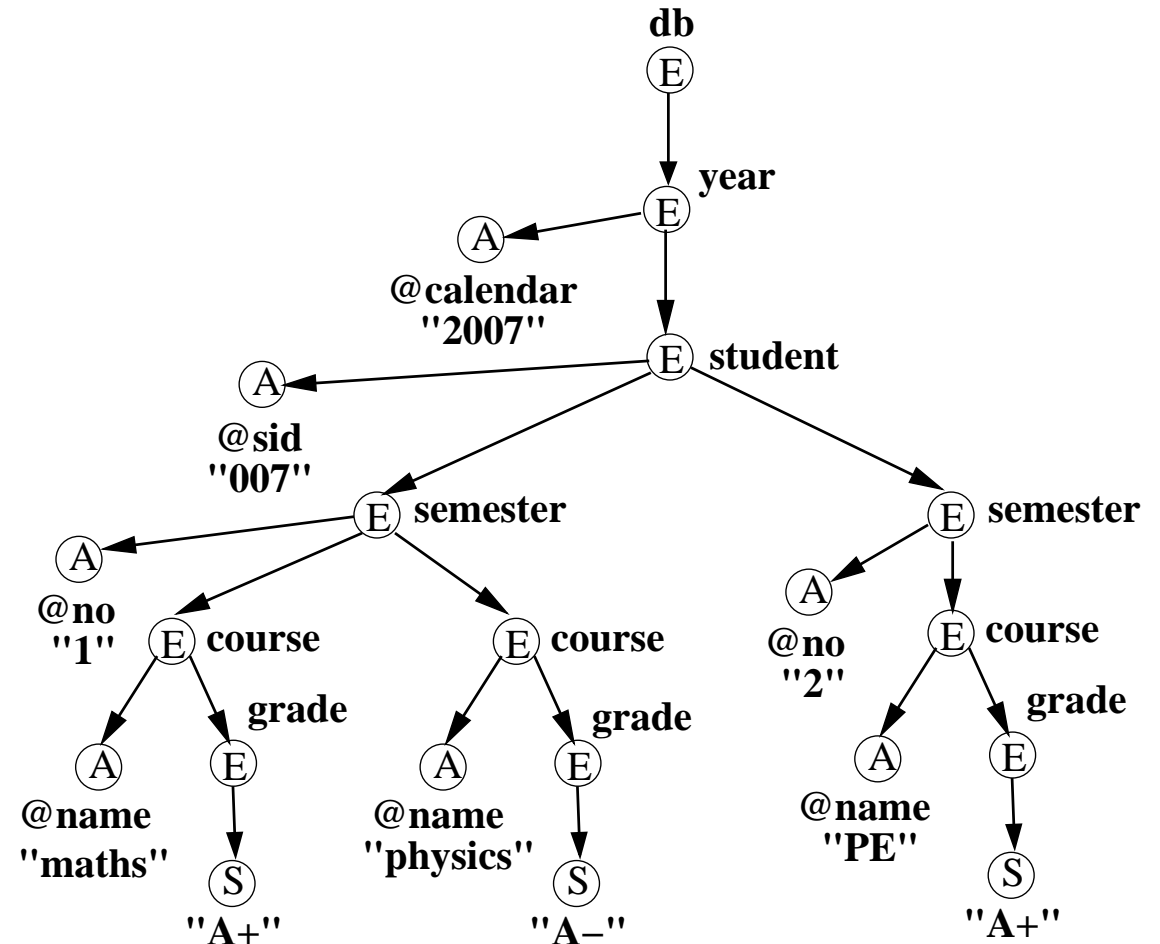
- ▶ Web data often encrypted (grades of students)
- ▶ following XQuery query retrieves grades of *Bart Simpson* in 2007:

```
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

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

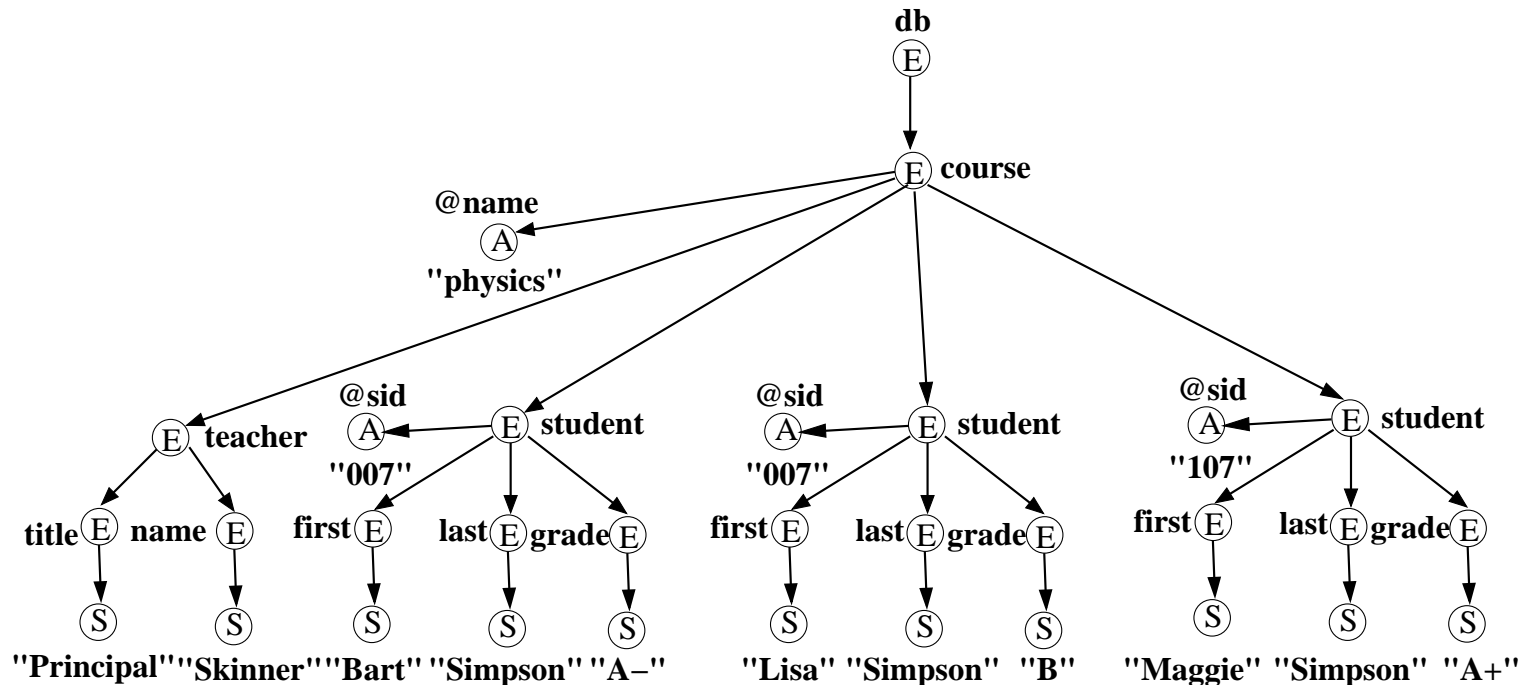
▶ as

```
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

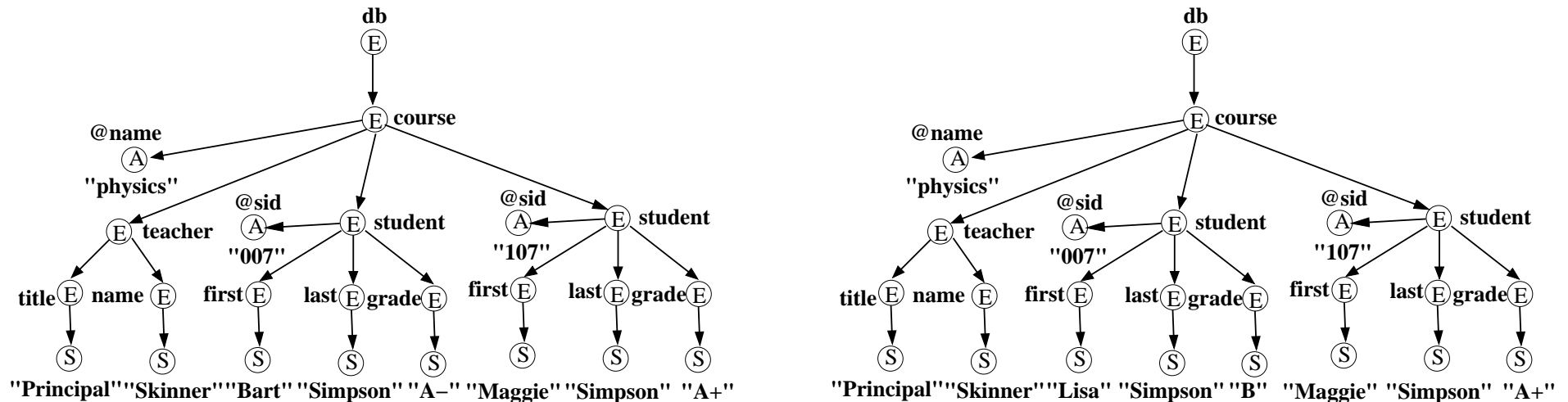
- ▶ approach to querying inconsistent databases without repairing them
- ▶ retrieves only certain answers, those present in *all* repairs



- ▶ *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:

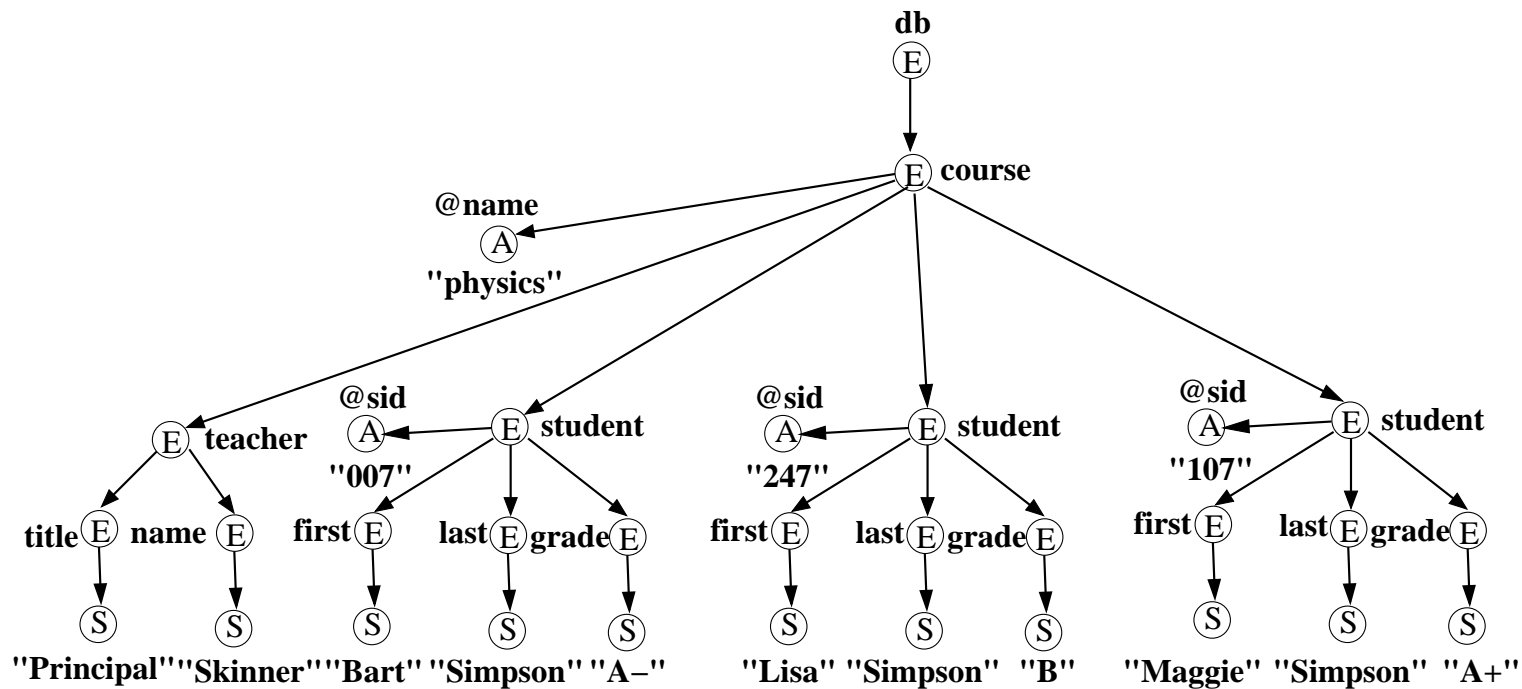


- ▶ CQA returns only *Maggie* as a certain answer:

`//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.:



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

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

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)

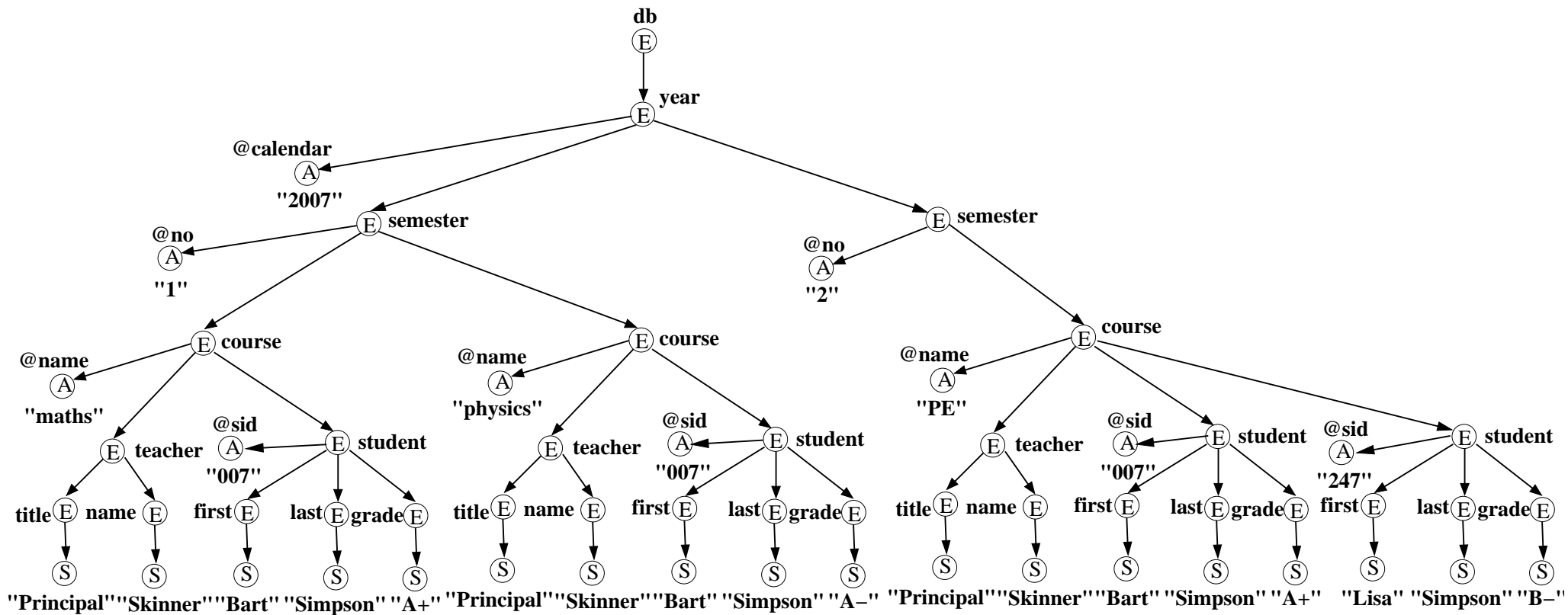
Fundamental Challenges and Outline

- ▶ 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 $T = (V, lab, ele, att, val, r)$
 - 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 $lab(v) \in \mathbf{A}$
 - element nodes v have element name $lab(v) \in \mathbf{E}$
 - text nodes v have fixed name $lab(v) = S$, root r is element node
- element nodes v have children
 - a set $att(v) \subseteq V$ of attribute nodes
 - a list $ele(v) \subseteq V$ 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 $val(v) \in \mathbf{String}$

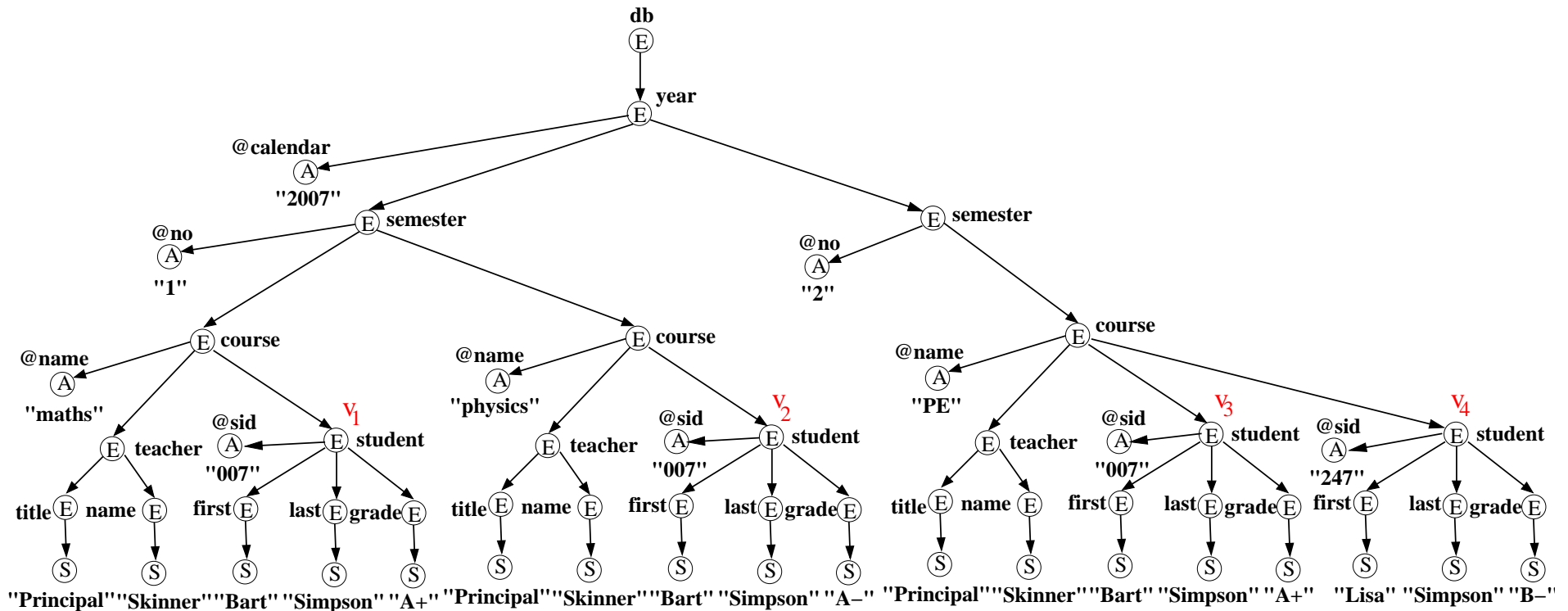
An Example of an XML Tree



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:
 - $lab(v_1) = lab(v_2)$
 - if v_1, v_2 are attribute or text nodes, then $val(v_1) = val(v_2)$
 - 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: $v_1 =_v v_3$, $v_2 \neq_v v_1$ and $v_2 \neq_v v_3$
- no two distinct *course*-nodes are value-equal
- for $V_1 = \{v_1, v_2\}$ and $V_2 = \{v_3, v_4\}$ we have $V_1 \cap_v V_2 = \{(v_1, v_3)\}$

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

<u>Path Language</u>	<u>Syntax</u>
PL_s	$P ::= \epsilon \mid P.l$
PL	$Q ::= \epsilon \mid Q.l \mid Q._*$

- $l \in \mathbf{E} \cup \mathbf{A} \cup \{S\}$ denotes any node name
- $_*$ denotes a *wildcard*, ϵ 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:

↳ year.semester

↳ semester.student

↳ year.semester.course.student.grade

▶ path expressions:

↳ $_{*}$.year.semester

↳ $_{*}$.course. $_{*}$.grade

▶ simple path expressions in the regular language of path expressions:

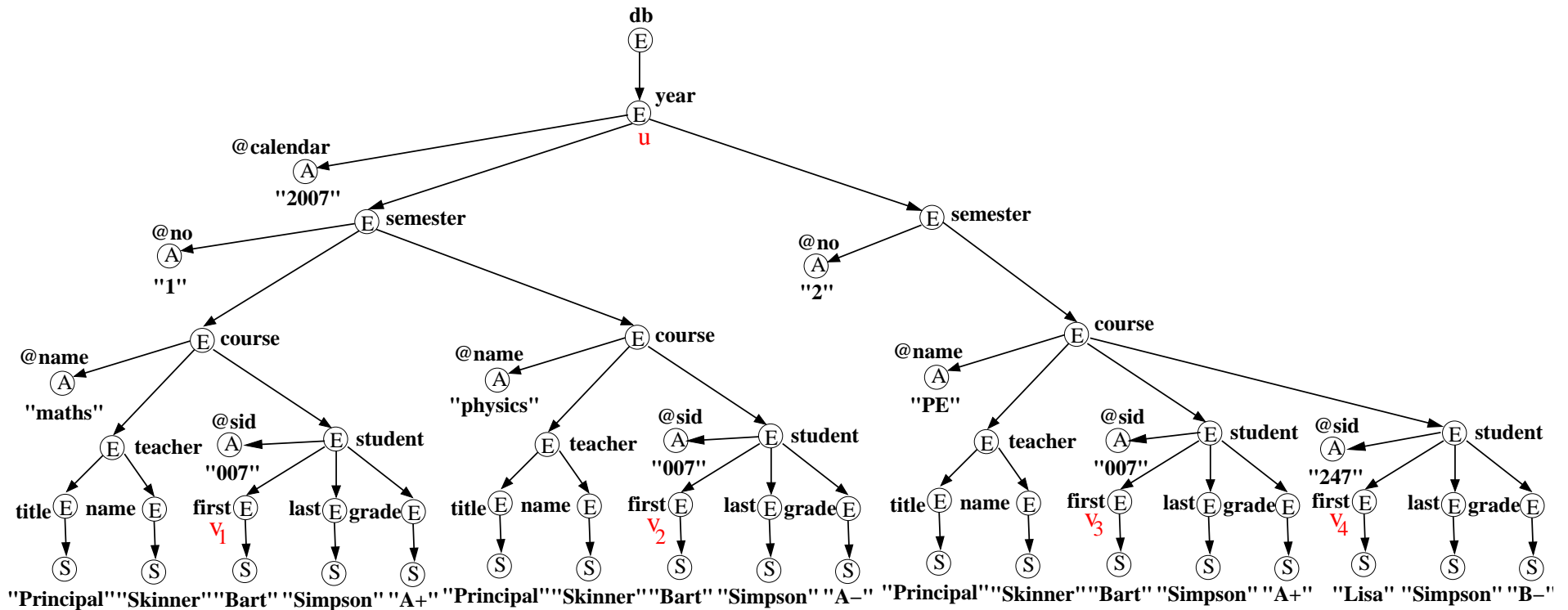
↳ year.semester \in $_{*}$.year.semester

↳ year.semester.course.student.grade \in $_{*}$.course. $_{*}$.grade

Selecting nodes in an XML tree

- path expression Q said to be *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 *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



- ▶ $u[-^*.course._^*.first] = \{v_1, v_2, v_3, v_4\}$
- ▶ $semester.course._^*.first \subseteq _^*.course._^*.first$
- ▶ $not _^*.course._^*.first \subseteq semester.course._^*.first$

The Path Containment Problem

- some results on the complexity of the path containment problem:

PTIME $XP(/, //), XP(/, //, []), XP(/, [], *), XP(/, //, [])$

$XP(/, //, *, DTD)$

coNP-complete $XP(/, //, [], *), XP(/, //, [], *, |), XP(/, [], DTD)$

coNP-hard $XP(/, |), XP(//, |), XP(//, [], DTD)$

EXPTIME-complete $XP(/, //, [], *, |, DTD)$

EXPTIME-hard $XP(/, //, [], *, DTD), XP(/, //, |, DTD)$

- M. Benedikt, W. Fan, and G. Kuper. *Structural properties of XPath fragments*. Theoretical Computer Science, 336(1):3–31, 2005.
- G. Miklau, D. Suciu. *Containment and equivalence for a fragment of XPath*. Journal of the ACM, 51(1):2–45, 2004.
- F. Neven, T. Schwentick. *XPath containment in the presence of disjunction, DTDs, and variables*. ICDT, PP. 315–329, 2003.

Cardinality constraints for XML

- a *cardinality constraint* φ has the form

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

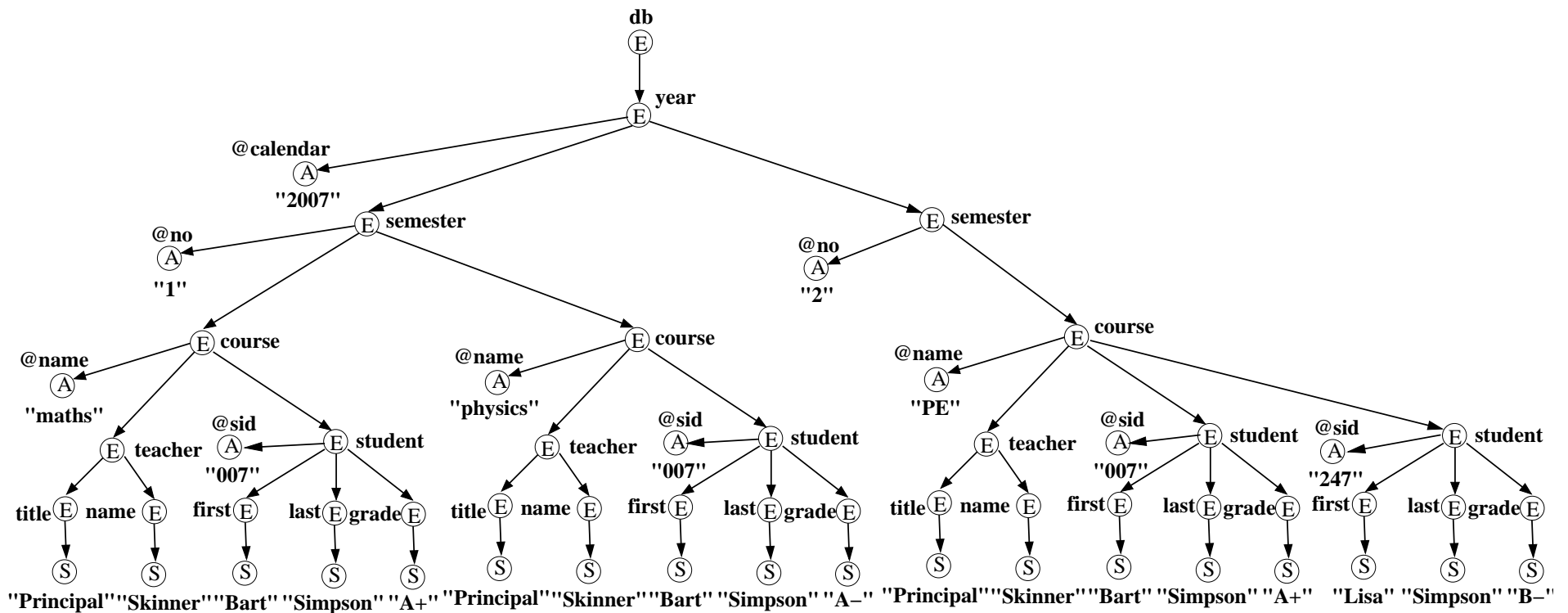
- $Q \in PL$ is the *context path*
- $Q' \in PL$ is the *target path*
- $Q_i \in PL$ is a *key path* (for $i = 1, \dots, k$)
- $Q.Q'$ valid ($k = 0$) or $Q.Q'.Q_i$ valid path (for $i = 1, \dots, k$)
- two nodes v_1, v_2 are $\{Q_1, \dots, Q_k\}$ -*confusable* iff

$$v_1[[Q_i]] \cap_v v_2[[Q_i]] \neq \emptyset \text{ for all } i = 1, \dots, k$$
- the XML tree T *satisfies* φ 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 $\{Q_1, \dots, Q_k\}$ -confusable with q'

Example 1

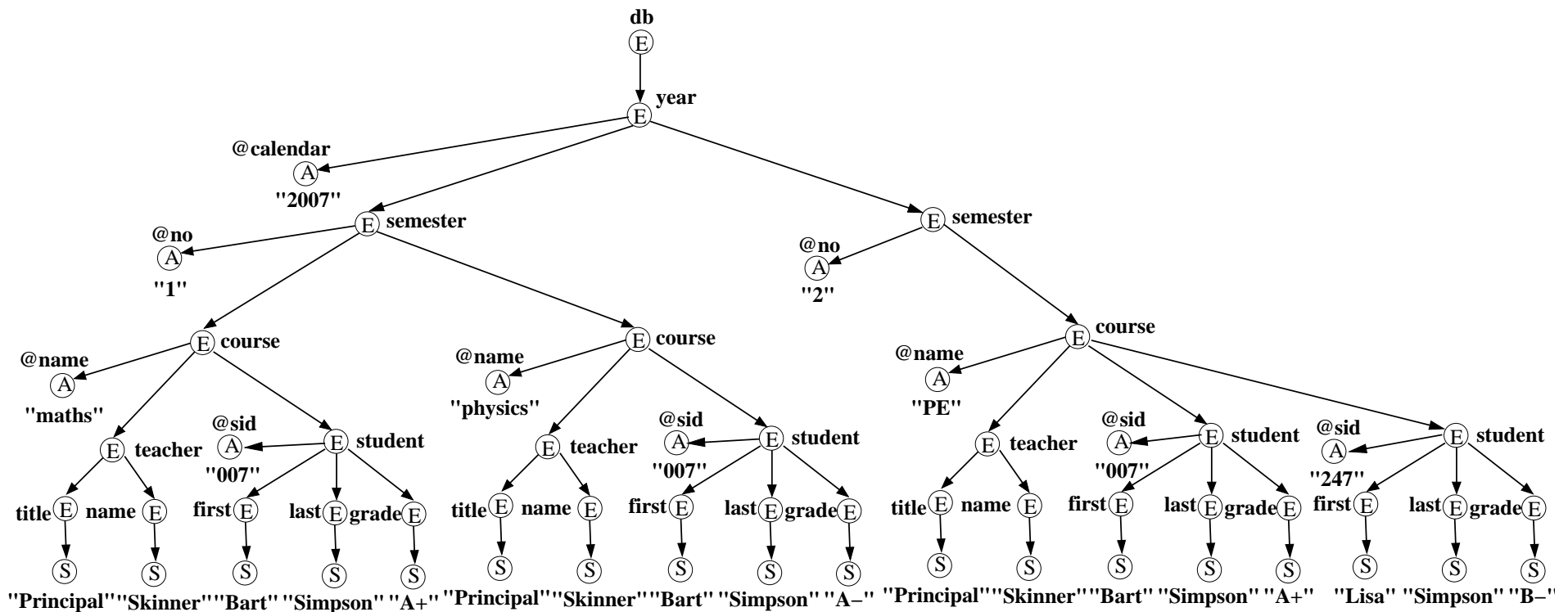
- ▶ in every semester every student who studies must enrol in 2 to 4 courses

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



Example 2

- ▶ T satisfies $card(year, (_*.course, \{teacher\})) = (3, 6)$
- ▶ T violates $card(_*.semester, (course, \{teacher\})) = (2, 2)$



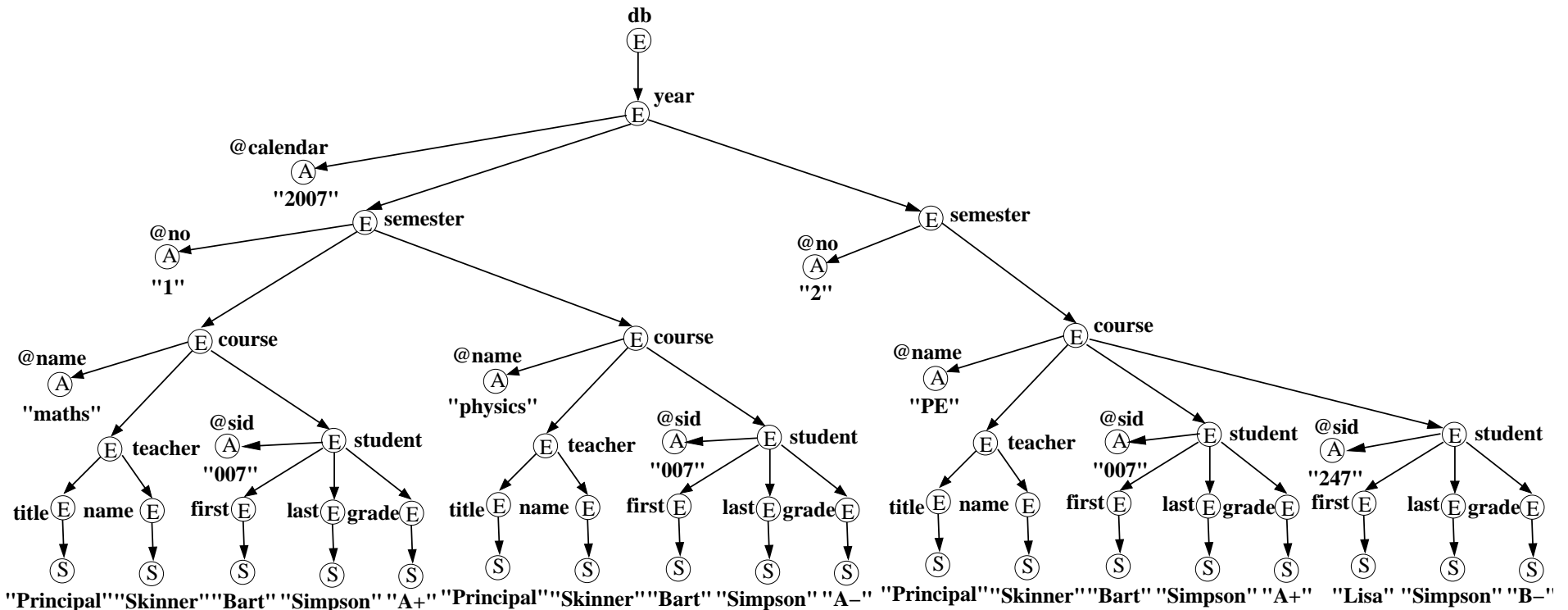
Example 3

- ▶ XML keys covered by cardinality constraints: $min = max = 1$

↳ P. Buneman et al. Reasoning about keys for XML. Inf. Syst.28(8):1037–1063, 2003

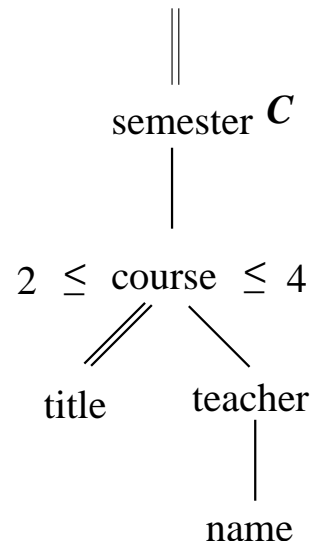
↳ S. Hartmann, S. Link. Efficient Reasoning about a Robust XML Key Fragment. TODS 34(2:10), 2009

- ▶ T satisfies $card(-*.course, (student, \{sid\})) = (1, 1)$



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: $card(-*.semester, (course, \{-*.title, teacher.name\})) = (2, 4)$
- ▶ visually as a *pattern*:



Decision Problems

- ▶ (finite) satisfiability problem:
 - ↪ Given any finite Σ , is there (finite) T satisfying all $\sigma \in \Sigma$?
- ▶ Σ (finitely) *implies* $\varphi \in \Sigma^*$ (*semantic closure*):
 - ↪ each (finite) XML tree that satisfies Σ also satisfies φ
- ▶ (finite) implication problem:
 - ↪ Given any finite $\Sigma \cup \{\varphi\}$: does Σ (finitely) imply φ ?
- ▶ efficient solution to decision problems unlocks XML applications
- ▶ let $\Sigma_{\mathcal{R}}^+$ be the *syntactic closure* of Σ
 - with respect to a fixed system \mathcal{R} of inference rules
 - $\varphi \in \Sigma_{\mathcal{R}}^+$ can be *inferred* from Σ using the rules in \mathcal{R}
- ▶ \mathcal{R} is *sound* iff $\Sigma_{\mathcal{R}}^+ \subseteq \Sigma^*$, and *complete* iff $\Sigma^* \subseteq \Sigma_{\mathcal{R}}^+$

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:

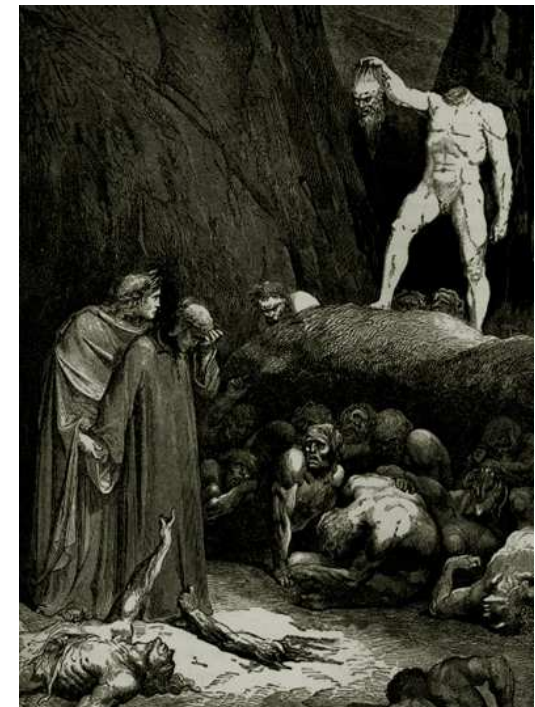
- ↳ restriction to tractable cases

- ↳ do penance by studying properties

- ▶ our "Divine Light":

- ↳ finite axiomatization

- ↳ efficient algorithms



Sources of Intractability: Lower and Upper Bounds

- ▶ $\mathcal{C}_1 = \{\text{card}(\epsilon, (P', \{P_1, \dots, P_k\})) = (\min, \max) \mid k \geq 1)\}$
- ▶ Theorem:
The finite implication problem for the class \mathcal{C}_1 of all simple absolute cardinality constraints with a non-empty set of key paths is *coNP*-hard.
- ▶ suggests that computational intractability may result from the specification of both lower and upper bounds

Sources of Intractability: Empty sets of key paths

▶ $\mathcal{C}_2 = \{\text{card}(\epsilon, (P', \{P_1, \dots, P_k\})) = (1, \max) \mid k \geq 0\}$

▶ Theorem:

The finite implication problem for the class \mathcal{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

▶ $\mathcal{C}_3 = \{\text{card}(\epsilon, (Q', \{Q_1, \dots, Q_k\})) = (1, \max) \mid k \geq 1\}$

▶ Theorem:

The finite implication problem for the class \mathcal{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* φ has the form

$$\text{card}(Q, (Q', \{P_1, \dots, P_k\})) \leq \text{max}$$

- $Q \in PL$ is the *context path*
 - $Q' \in PL$ is the *target path*
 - $P_i \in PL$ is a *key path* (for $i = 1, \dots, k$ and $k \geq 1$)
 - $Q.Q'.P_i$ valid path (for $i = 1, \dots, k$)
- ▶ XML tree T satisfies $\text{card}(Q, (Q', \{P_1, \dots, P_k\})) \leq \text{max}$ iff T satisfies $\text{card}(Q, (Q', \{P_1, \dots, P_k\})) = (1, \text{max})$
- ▶ 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

$$\frac{}{card(Q, (Q', S)) \leq \infty}$$

(infinity)

$$\frac{}{card(Q, (\epsilon, S)) \leq 1}$$

(epsilon)

$$\frac{card(Q, (Q', S)) \leq max}{card(Q, (Q', S)) \leq max + 1}$$

(weakening)

$$\frac{card(Q, (Q', S)) \leq max}{card(Q, (Q', S \cup \{P\})) \leq max}$$

(superkey)

$$\frac{card(Q, (Q'.P, \{P'\})) \leq max}{card(Q, (Q', \{P.P'\})) \leq max}$$

(subnodes)

$$\frac{card(Q, (Q'.Q'', S)) \leq max}{card(Q.Q', (Q'', S)) \leq max}$$

(context target)

$$\frac{card(Q, (Q', S)) \leq max}{card(Q'', (Q', S)) \leq max}^{Q'' \subseteq Q}$$

(context-path-containment)

$$\frac{card(Q, (Q'.P, \{\epsilon, P'\})) \leq max}{card(Q, (Q', \{\epsilon, P.P'\})) \leq max}$$

(subnodes-epsilon)

$$\frac{card(Q, (Q', S)) \leq max}{card(Q, (Q'', S)) \leq max}^{Q'' \subseteq Q'}$$

(target-path-containment)

$$\frac{card(Q, (Q', S \cup \{\epsilon, P\})) \leq max}{card(Q, (Q', S \cup \{\epsilon, P.P'\})) \leq max}$$

(prefix-epsilon)

$$\frac{card(Q, (Q', \{P.P_1, \dots, P.P_k\})) \leq max, \quad card(Q.Q', (P, \{P_1, \dots, P_k\})) \leq max'}{card(Q, (Q'.P, \{P_1, \dots, P_k\})) \leq max \cdot max'}$$

(multiplication)

Example

- ▶ sports league competition:
- ▶ per season same home and away team both play in at most 3 months:

$$\sigma_1 = \text{card}(_*.season, (\text{month}, \{\text{match.home}, \text{match.away}\})) \leq 3$$

- ▶ per month same home team plays same away team at most twice:

$$\sigma_2 = \text{card}(_*.season.month, (\text{match}, \{\text{home}, \text{away}\})) \leq 2$$

- ▶ same home team plays same away team at most 5 times per season :

$$\varphi = \text{card}(_*.season, (\text{month.match}, \{\text{home}, \text{away}\})) \leq 5$$

- ▶ φ cannot be inferred from $\Sigma = \{\sigma_1, \sigma_2\}$

↪ How can we show that φ is not implied by Σ ?

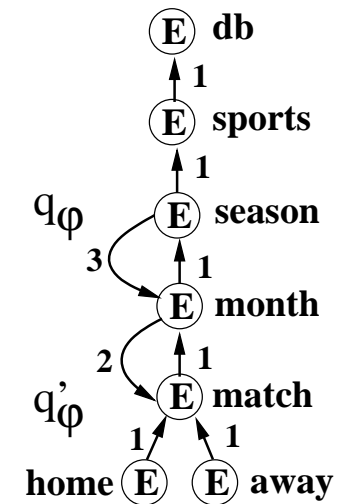
↪ we construct a counter-example

Shortest Paths vs Derivability

- ▶ we construct a *cardinality graph* $G_{\Sigma, \varphi}$ from Σ and φ
- ▶ if $d(q_\varphi, q'_\varphi) \leq \max_\varphi$ in $G_{\Sigma, \varphi}$, then $\varphi \in \Sigma^+$
- ▶ in other words:
 - if φ cannot be inferred from Σ , then $d(q_\varphi, q'_\varphi) \geq \max_\varphi + 1$ in $G_{\Sigma, \varphi}$

▶ Example:

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



Completeness

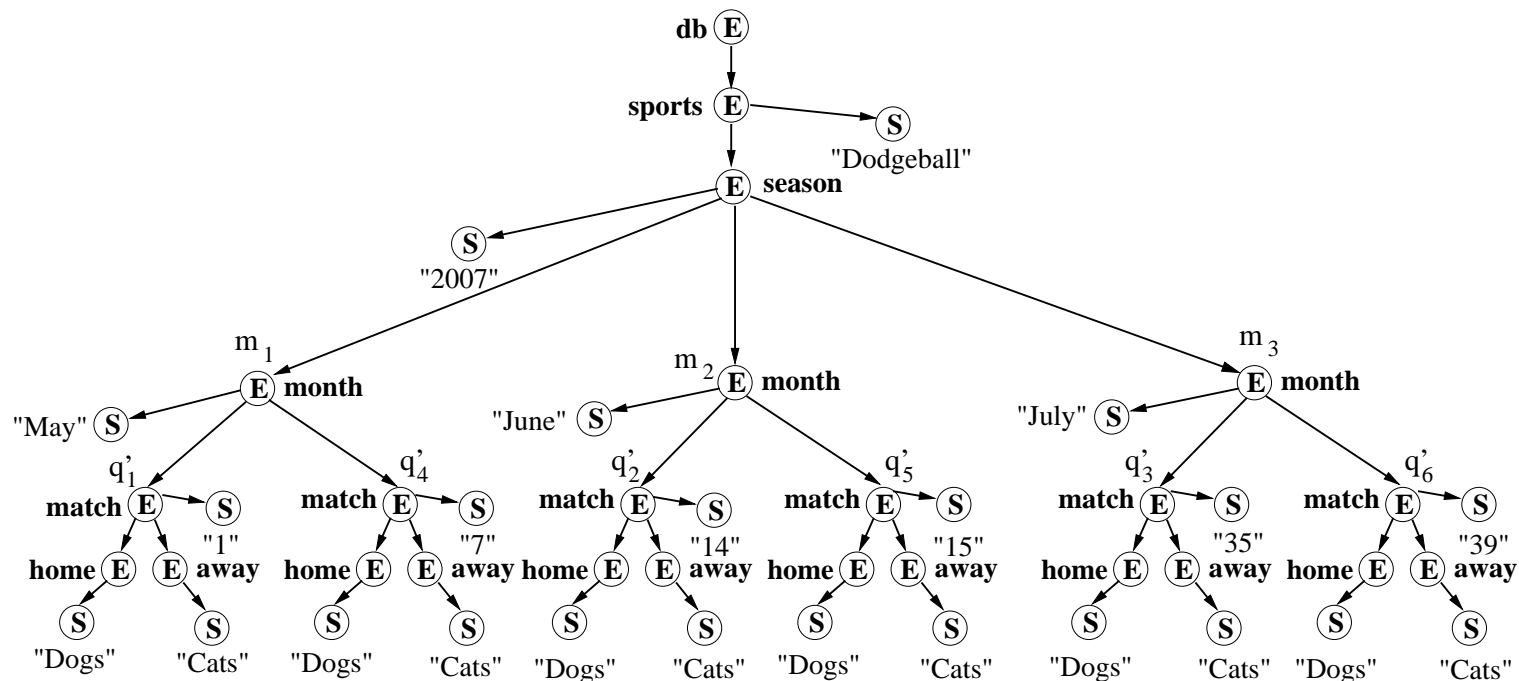
▶ T satisfies

$$\hookrightarrow \sigma_1 = \text{card}(-^*.season, (\text{month}, \{\text{match.home}, \text{match.away}\})) \leq 3$$

$$\hookrightarrow \sigma_2 = \text{card}(-^*.season.month, (\text{match}, \{\text{home}, \text{away}\})) \leq 2$$

▶ T violates

$$\hookrightarrow \varphi = \text{card}(-^*.season, (\text{month.match}, \{\text{home}, \text{away}\})) \leq 5$$



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:** Σ, φ
 - **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_φ ($\mathcal{O}(|\varphi|^2)$)
- cardinality graph constructable in time $\mathcal{O}(|\Sigma| \cdot |\varphi|^3)$
 - we use the evaluation algorithm for Core XPath queries
 - G. Gottlob, Ch. Koch, R. Pichler. *Efficient Algorithms for Processing XPath Queries*. TODS, 2005.
- in fact: cardinality graph constructable in time $\mathcal{O}(|\Sigma| \cdot |\varphi|)$

A finite axiomatisation for XML keys

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

$$\frac{}{(Q, (\epsilon, S))} \quad \text{(epsilon)}$$

$$\frac{(Q, (Q', S \cup \{\epsilon, P\}))}{(Q, (Q', S \cup \{\epsilon, P.P'\}))} \quad \text{(prefix-epsilon)}$$

$$\frac{(Q, (Q', S))}{(Q, (Q', S \cup \{P\}))} \quad \text{(superkey)}$$

$$\frac{(Q, (Q'.P, \{P'\}))}{(Q, (Q', \{P.P'\}))} \quad \text{(subnodes)}$$

$$\frac{(Q, (Q', S))}{(Q'', (Q', S))} \quad Q'' \subseteq Q \quad \text{(context-path-containment)}$$

$$\frac{(Q, (Q', S))}{(Q, (Q'', S))} \quad Q'' \subseteq Q' \quad \text{(target-path-containment)}$$

$$\frac{(Q, (Q'.Q'', S))}{(Q.Q', (Q'', S))} \quad \text{(context target)}$$

$$\frac{(Q, (Q'.P, \{\epsilon, P'\}))}{(Q, (Q', \{\epsilon, P.P'\}))} \quad \text{(subnodes-epsilon)}$$

$$\frac{(Q, (Q', \{P.P_1, \dots, P.P_k\})), (Q.Q', (P, \{P_1, \dots, P_k\}))}{(Q, (Q'.P, \{P_1, \dots, P_k\}))} \quad \text{(interaction)}$$

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:** Σ, φ
 - **Output:** YES if $\varphi \in \Sigma^*$; No otherwise
 - **Method:**
 - (1) construct the reachability graph $G_{\Sigma, \varphi}$
 - (2) **if** q'_φ reachable from q_φ **then return**(YES) **else return**(NO)
- reachability test: depth-first search of $G_{\Sigma, \varphi}$ with root q_φ in $\mathcal{O}(|\varphi|^2)$
- reachability graph constructable in time $\mathcal{O}(|\Sigma| \cdot |\varphi|)$
 - we use the evaluation algorithm for Core XPath queries
 - G. Gottlob, Ch. Koch, R. Pichler. *Efficient Algorithms for Processing XPath Queries*. TODS, 2005.

Structural XML Keys

- keys: target nodes uniquely identified by values reachable via key paths
- structural keys identify target nodes independently of any data values
- $(_{-}^*.course, (name, \emptyset))$, but not $(_{-}^*.semester, (_{ -}^*.name, \emptyset))$
- allow structural keys: XML key is expression $(Q, (Q', S))$ where
 - Q, Q' are PL expressions,
 - S is a finite set of PL_S 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

$$\begin{array}{c}
 \frac{}{(Q, (\varepsilon, S))} \\
 \text{(epsilon)}
 \end{array}
 \qquad
 \frac{(Q, (Q', S \cup \{P, P.P'\})), (Q.Q', P, \emptyset)}{(Q, (Q', S \cup \{P, P.P'.P''\}))} \\
 \text{(prefix)}
 \end{array}
 \qquad
 \frac{(Q, (Q', S))}{(Q, (Q', S \cup \{P\}))} \\
 \text{(superkey)}
 \end{array}$$

$$\frac{(Q, (Q'.P, \{P'\}))}{(Q, (Q', \{P.P'\}))} \\
 \text{(subnodes)}
 \end{array}
 \qquad
 \frac{(Q, (Q', S))}{(Q'', (Q', S))}^{Q'' \subseteq Q} \\
 \text{(context-path-containment)}
 \end{array}
 \qquad
 \frac{(Q, (Q', S))}{(Q, (Q'', S))}^{Q'' \subseteq Q'} \\
 \text{(target-path-containment)}
 \end{array}$$

$$\frac{(Q, (Q'.Q'', S))}{(Q.Q', (Q'', S))} \\
 \text{(target-to-context)}
 \end{array}
 \qquad
 \frac{(Q, (Q'.P, \{\varepsilon, P'\}))}{(Q, (Q', \{\varepsilon, P.P'\}))} \\
 \text{(subnodes-epsilon)}
 \end{array}
 \qquad
 \frac{(Q, (Q', \{P.P' \mid P' \in S\})), (Q.Q', (P, S))}{(Q, (Q'.P, S))} \\
 \text{(interaction)}
 \end{array}$$

$$\frac{(Q.Q', (Q'', S)), (Q, Q', \emptyset)}{(Q, (Q'.Q'', S))} \\
 \text{(context-to-target)}
 \end{array}
 \qquad
 \frac{(Q, (-*.P, \emptyset))}{(\varepsilon, (Q.-*.Q'.-*, \emptyset))}^{Q' \neq *} \\
 \text{has proper border } P \\
 \text{(border)}
 \end{array}
 \qquad
 \frac{(Q, (Q'.P, \{P_1, \dots, P_k\})), (Q.Q', (P, \emptyset))}{(Q, (Q', \{P.P_1, \dots, P.P_k\}))} \\
 \text{(multiple subnodes)}
 \end{array}$$

$$\frac{(Q, (-*.P, \emptyset))}{(\varepsilon, (Q.-*, \{P_1.P.P', P_2.P.P''\}))}^{P_1 \neq P_2} \\
 \text{(neighbour)}
 \end{array}$$

Deciding Implication

- method may fail: reachability graph can violate structural key
- pre-processing: merge distinct key paths to satisfy structural keys
 - **Input:** Σ, φ
 - **Output:** YES if $\varphi \in \Sigma^*$; No otherwise
 - **Method:**
 - (1) Construct $\bar{T}_{\Sigma, \varphi}$ from Σ and φ ;
 - (2) **IF** $\exists(Q_\sigma, (Q'_\sigma, \emptyset)) \in \Sigma. \exists w_\sigma \in [Q_\sigma]$ in $\bar{T}_{\Sigma, \varphi}$
and $\exists w'_\sigma, w''_\sigma \in w_\sigma[Q'_\sigma]$ in $\bar{T}_{\Sigma, \varphi}$ with $w'_\sigma \neq w''_\sigma$
 - (3) **THEN RETURN(yes)**
 - (4) **ELSE** Construct $G_{\Sigma, \varphi}$ from Σ and φ ;
 - (5) **IF** q_φ is reachable from q'_φ in $G_{\Sigma, \varphi}$ **THEN RETURN(yes)**
 - (6) **ELSE RETURN(no)**.
- implication still decidable in time $\mathcal{O}(|\varphi| \times (||\Sigma|| + |\varphi|))$

Findings for Cardinality Constraints

- ▶ form 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 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?

