RADICAL 2010
RADICAL 2010

Wiki
Everyone can use to
Store,
Organize,
Manage and
Exchange data
RADICAL 2010

Wiki

Everyone can use to

Store,

Organize,

Manage and

Exchange data

James Cheney
University of Edinburgh

Joint work with
Peter Buneman, Sam
Lindley, Heiko Mueller
(UoE)
Michael Benedikt (Oxford)
Or:

*How to train your database wiki*
Or:

How to train your database wiki
Or:

This is what happens when James has too much coffee
Curated databases

• Created by manual effort
• Curators copy data from papers, other DBs
• Some sources unreliable
  • some curators too
Curated databases

- Created by manual effort
- Curators copy data from papers, other DBs
- Some sources unreliable
  - some curators too

Hi, everybody!
Problem

• We know basically what to do
• "Curated databases" should provide built-in:
  • archiving [Buneman et al. 04, 08, ...]
  • provenance [Buneman et al. 01, 06, 07, 08, ...]
  • annotation [Geerts et al. 06, ...]
• But hard to convince users to do the right thing
Solution:
A database wiki

- Standard wiki stuff
  - WikiLinks, editable pages, brain-dead syntax, page history
- New: editable, (semi)structured data with
  - Transclusion (via queries embedded in pages)
  - Annotation (discuss data, propose changes)
  - Stable citation, copy/paste for wiki data
  - Archiving - record all past versions
  - Provenance - automagical (?)
Implementation

• Using Links
  • Other web programming languages would probably also work (but we have in-house expertise)

• Currently:
  • basic wiki stuff
  • "data tree" - editable via browser & persistent
  • path transclusions and type-based selection queries
and now for something completely different:

Independence Analysis for semistructured data
Motivation

• Suppose we have multiple (cached) pages expressed by (XQuery/XPath) queries $Q_1, Q_2, \ldots$

• When the database wiki is updated:
  • Which queries may be affected?
  • If we can determine (quickly) that queries and updates are independent
  • then can keep using cached version of unchanged pages
Query-update independence
Query-update independence
Query-update independence

- DB
- Q
- Q(DB)
- U(DB)
- U
Query-update independence
Query-update independence

\[ \text{Q}(\text{U(DB)}) = ??? \]
Independence example

```
for $x$ in /c/a
return <d>$x</d>
```
Independence example

for $x$ in /c/a
    return <d>$x</d>
Independence example

for $x$ in /c/a
return <d>$x</d>
Independence example

```
// Independence example

for $x$ in /c/a
    return <d>$x$</d>
```

```
// Independence example

for $x$ in /c/a
    return <d>$x$</d>
```

```
// Independence example

for $x$ in /c/a
    return <d>$x$</d>
```
Independence example

for $x$ in /c/a
return <d>$x</d>

for $x$ in /c/a
return <d>$x</d>

=
Independence non-example

for $x$ in /a
return <d>$x</d>
Independence **non-example**
Independence **non-example**

for $x$ in /a
return <d>$x</d>

for $x$ in /a
return <d>$x</d>

≠
Prior work

• [Benedikt and C 09]: static independence analysis based on schema (reg. exp. types)
  • Showed effective for avoiding view maintenance

• Problem: Useless if you don't have a schema
  • Some prior work on path-based "XML projection" [Marian & Simeon 03,...], "commutativity analysis" [Ghelli et al. 08]
  • But doesn't quite solve independence problem
Current work

• [Benedikt and C 2010]: Use queries to statically describe the set of updates that "may destabilize" the query

• Call this $\Delta(Q)$, the destabilizer (or antiprovenance) of $Q$

• $\text{Targets}(U)$ disjoint from $\Delta(Q)$ implies $Q$ independent of $U$

• Can be just as effective, without a schema
Key subproblem: XPath intersection analysis

• In the absence of schema, use paths to statically describe sets of nodes
  • cf. [Ghelli, Simeon & Rose 2008], others

• Intersection of downward paths is $O(n^2)$

• But general problem is NP-hard
Solvers to the rescue?

- There are solvers for decidable tree logics
- MONA (decides MSO(Tree))
  - Somewhat unpredictable, needs tuning
- [Geneves et al. 07]: Modal mu-calc solver
  - Source not available
  - Optimized version not yet available
A special case

• Our approach: novel (apparently) reduction from EFO(Tree) to EFO(N, <).

• Most SMT solvers are very good at EFO(N,<)-SAT (typically complete)

• Typically faster than MONA or Geneves solver on our path intersection/independence benchmark

• but handles a much weaker theory
Idea

- Sibling(x, y)
- \( \Rightarrow \)
- \( x.\text{post} < y.\text{pre} \)
Idea

- $\text{Desc}(x,y)$
- $\Rightarrow$
- $x.\text{pre} < y.\text{pre} \& x.\text{post} > y.\text{post}$

1 2 3 4 5 6 7 8 9 10
Child(x, y) =>

Desc(x, y) & not (Desc(x, x1) & Desc(x1, y))
& ...
& not (Desc(x, xn) & Desc(xn, y))
Comparison for one typical problem

(not a comprehensive experiment!)
Experimental results

• Destabilizer-based independence analysis is just as effective as schema-based
  • succeeds/fails on different queries
  • fast enough (using yices) to yield savings

• Close to exact
  • < 1% false positives on benchmark of over 500 problems
  • (hand classified)
Respectable charts and figures

Figure 5: Running times for the generic analysis, in milliseconds (logarithmic scale), broken down by update and analysis level.

Effectiveness of generic analyses

Figure 6: Effectiveness of the generic analysis, expressed as a percentage of query-update pairs determined independent, broken down by update and by analysis level.

<table>
<thead>
<tr>
<th></th>
<th>SCH</th>
<th>L1</th>
<th>L2</th>
<th>L2 + SCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1MB</td>
<td>8.3%</td>
<td>10.5%</td>
<td>8.3%</td>
<td>10.9%</td>
</tr>
<tr>
<td>2.3MB</td>
<td>11.0%</td>
<td>14.5%</td>
<td>14.9%</td>
<td>20.1%</td>
</tr>
</tbody>
</table>

Table 1: Maintenance time savings across whole benchmark
Respectable charts and figures

Figure 5: Running times for the generic analysis, in milliseconds (logarithmic scale), broken down by update and analysis level.

Figure 6: Effectiveness of the generic analysis, expressed as a percentage breakdown by update and by analysis level.

<table>
<thead>
<tr>
<th></th>
<th>SCH</th>
<th>L1</th>
<th>L2</th>
<th>L2 + SCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1MB</td>
<td>8.3%</td>
<td>10.5%</td>
<td>8.3%</td>
<td>10.9%</td>
</tr>
<tr>
<td>2.3MB</td>
<td>11.0%</td>
<td>14.5%</td>
<td>14.9%</td>
<td>20.1%</td>
</tr>
</tbody>
</table>

Table 1: Maintenance time savings across whole benchmark
Limitations/future work

• What if you do have a schema?
  • Naive joint path and schema analysis works OK
  • Smarter: destabilizer intersection modulo schema

• Schemas not expressible exactly in EFO(N)
  • Can SMT approach be extended to handle schemas?

• What about incremental maintenance?
  • ideally, want to combine static and dynamic approaches
Conclusions

• Database wikis will be A.W.E.S.O.M.E.
• and will need good high-level web, XML and constraint programming tools
• Need to solve tree constraints quickly using SMT solvers (or other decision procedures?)
• in order to make R.A.D.I.C.A.L. advances in techniques for database curation