A graph model for data and workflow provenance

Umut Acar, Peter Buneman, James Cheney, Natalia Kwasnikowska, Jan van den Bussche, & Stijn Vansummeren

TaPP 2010
Provenance in ...

- Databases
  - Mainly for (nested) relational model
  - Where-provenance ("source location")
  - Lineage, why ("witnesses")
  - How/semiring model
  - Relatively formal

- Workflows
  - Many different systems
  - Many different models
    - (converging on OPM?)
  - Graphs/DAGs
  - Relatively informal
Provenance in ...

- **Databases**
  - Mainly for (nested) relational model
  - Where-provenance ("source location")
  - Lineage, why ("witnesses")
  - How/semiring model
  - Relatively formal

- **Workflows**
  - Many different systems
  - Many different models
  - Graphs/DAGs
  - Relatively informal
This talk

• Relate database & workflow "styles"
• Develop a common graph formalism
• Need a common, expressive language that
  • supports many database queries
  • describes some (simple) workflows
Previous work

- Dataflow calculus (DFL), based on nested relational calculus (NRC)
  - Provenance "run" model by Kwasnikowska & Van den Bussche (DILS 07, IPAW 08)
- "Provenance trace" model for NRC
  - by (Acar, Ahmed & C. '08)
- Open Provenance Model (bipartite graphs)
  - (Moreau et al. 2008-9), used in many WF systems
NRC/DFL background

- A very simple, functional language:
  - basic functions +, *, ... & constants 0, 1, 2, 3...
  - variables x, y, z
  - pair/record types (A:e,...,B:e), \( \pi_A(e) \)
  - collection (set) types
    - \{e,...\} \( e \cup e \) \{e \mid x \text{ in } e'\} \cup e
An example
An example

• Suppose \( R = \{(1,2,3), (4,5,6), (9,8,7)\} \)
An example

- Suppose $R = \{(1, 2, 3), (4, 5, 6), (9, 8, 7)\}$

\[
\text{sum}\ \{\ x \times y \mid (x, y, z) \in R, \ x < y \}\n\]
An example

• Suppose $R = \{(1,2,3), (4,5,6), (9,8,7)\}$

$$\sum \{ x \cdot y \mid (x,y,z) \text{ in } R, x < y \}$$

$$= \sum \{ x \cdot y \mid (x,y,z) \text{ in } \{(1,2,3), (4,5,6)\} \}$$
An example

• Suppose \( R = \{(1,2,3), (4,5,6), (9,8,7)\} \)

\[
\begin{align*}
\text{sum } \{ x \times y \mid (x,y,z) \in R, x < y \} \\
= \text{sum } \{ x \times y \mid (x,y,z) \in \{(1,2,3), (4,5,6)\} \} \\
= \text{sum } \{1 \times 2, 4 \times 5\}
\end{align*}
\]
An example

• Suppose \( R = \{(1,2,3), (4,5,6), (9,8,7)\} \)

\[
\text{sum \{ x \times y \mid (x,y,z) \in R, \ x < y \}} \\
= \text{sum \{ x \times y \mid (x,y,z) \in \{(1,2,3), (4,5,6)\}\}} \\
= \text{sum \{1 \times 2, 4 \times 5\}} \\
= \text{sum \{2,20\}}
\]
An example

- Suppose \( R = \{(1,2,3), (4,5,6), (9,8,7)\} \)

\[
\text{sum } \{ x \cdot y \mid (x,y,z) \text{ in } R, \ x < y \}
\]
\[
= \text{sum } \{ x \cdot y \mid (x,y,z) \text{ in } \{(1,2,3), (4,5,6)\}\}
\]
\[
= \text{sum } \{1 \cdot 2, 4 \cdot 5\}
\]
\[
= \text{sum } \{2,20\}
\]
\[
= 22
\]
Another example

- In DFL, built-in functions / constants can be whole programs & files,

  - as in Provenance Challenge 1 workflow:

    ```
    let WarpParams := \{align_warp(img,hdr)
    | (img,hdr) in Inputs\} in
    let Reslices := \{reslice(wp)
    | wp in WarpParams\} in
    softmean(Reslices)
    ```
Goal: Define "provenance graphs" for DFL
Goal: Define "provenance graphs" for DFL

let WarpParams := {align_warp(img,hdr)}
| (img,hdr) in Inputs} in
let Reslices := {reslice(wp)
| wp in WarpParams} in
in softmean(Reslices)
Goal: Define "provenance graphs" for DFL

let WarpParams := {align_warp(img,hdr)
  | (img,hdr) in Inputs} in
let Reslices := {reslice(wp)
  | wp in WarpParams} in
in softmean(Reslices)

http://www.flickr.com/photos/schneertz/679692806/
First step: values

\[ \text{v} \rightarrow \text{c} \quad \text{or} \quad \text{v} \rightarrow \text{copy} \]

\[ \text{or} \quad \vdots \quad \text{or} \quad \vdots \]

\[ \text{or} \quad \{\} \quad \text{or} \quad \langle\rangle \]

\[ \text{v} \quad \text{v} \quad \text{v} \quad \text{v} \]

\[ \text{elem} \quad \text{elem} \quad A_i \quad A_n \]
Example value
Next step: evaluation nodes ("process")

- Constants, primitive functions
- Variables & temporary bindings
Pairing

Record building

Field lookup
Conditionals

Note: Only taken branch is recorded
Sets: basic operations

Empty set

Singleton

Union
Sets: complex operations

Flattening

Iteration
Provenance graphs

- are graphs with "both value and evaluation structure"
A bigger example
Value structure
Value structure
Return value
Expression structure
Expression structure
Building provenance graphs

• is complicated

• Here we'll use high-level "graph rewrite rule" formalism

• Mostly because it is nicer to look at than formal version
OK, take a deep breath!
An example
An example
An example

```
x, c
```

```
for x
```

```
head elem
```

```
body elem
```

```
+ elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```

```
{} elem
```
An example
An example
An example
An example
Graphs can "lie" (inconsistency)
Graphs can "lie"
(inconsistency)
Graphs can "lie" (inconsistency)
Graphs can "lie" (inconsistency)

"Locally" but not "globally" consistent
Graph queries

• Many possible approaches

• In paper: some Datalog
  • Maybe overkill, seems fragile

• In code: some "annotation propagation" traversals
  • Seems to handle where, "explanations", "summaries"
Explaining
Explaining
Explaining
Explaining

Note: Smallest consistent subgraph (NOT transitive closure!)
Summarizing
Graphs are partially "replayable"

- If we change a value node, can try to "readjust" to recover consistency

Formalized in (Acar, Ahmed, Cheney 08)
Graphs are partially "replayable"

- If we change a value node, can try to "readjust" to recover consistency

- Formalized in (Acar, Ahmed, Cheney 08)
Graphs are partially "replayable"

- If we change a value node, can try to "readjust" to recover consistency

- Formulated in (Acar, Ahmed, Cheney 08)
Graphs are partially "replayable"

• If we change a value node, can try to "readjust" to recover consistency

• Formalized in (Acar, Ahmed, Cheney 08)
Graphs are partially "replayable"

- If we change a value node, can try to "readjust" to recover consistency

- Formalized in (Acar, Ahmed, Cheney 08)
Graphs are partially "replayable"

• If we change a value node, can try to "readjust" to recover consistency

• Formalized in (Acar, Ahmed, Cheney 08)
Implementation in Haskell

- Summarized in paper, full code on request
  - roughly 250 LOC for basic evaluator
  - another 300 for graphviz translation, basic queries, examples
- Point?
  - No claim of efficiency/scalability but easy to understand, experiment
  - Elucidates some tricky details that pictures hide
  - Similar "lightweight modeling" might be valuable for understanding/relating other WF/DB models
Related work

• This work synthesizes/rearranges ideas from several previous works & "folklore"
  • traces (Acar, Ahmed, Cheney 2008)
  • runs (Kwasnikowska, van den Bussche, DILS 2007, IPAW 2008)
  • OPM graphs (Moreau et al. IPAW 2008 etc.)
    • and many workflow systems
• More can be done to relate DB & workflow models
Future work

• This is work in progress

• Next steps:
  • Extending to understand/model other workflow features
  • Better grasp of "real" queries and features needed
  • Implementa(tion|ability)?
  • Optimization?
Conclusions

- DB & WF provenance have much in common
- We develop common graph model
  - with both intuitive & precise presentations
- Still much to do to relate and integrate DB & WF models
  - let alone integrate models at scale in real systems