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

- Converging on OPM?
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 \((A:e,\ldots,B:e)\), \(\pi_A(e)\)

• collection (set) types
  • \{e,\ldots\} \quad e \cup e \quad \{e \mid x \text{ in } e'\} \quad \cup e
An example
An example

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

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

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

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

\[
\sum \{ x \times y \mid (x, y, z) \in R, x < y \} \\
= \sum \{ x \times y \mid (x, y, z) \in \{(1, 2, 3), (4, 5, 6)\} \}
\]
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 \( R = \{(1,2,3), (4,5,6), (9,8,7)\} \)

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

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

\[
\begin{align*}
\text{sum } \{ x \times y \mid (x,y,z) \text{ in } R, x < y \} &= \text{sum } \{ x \times y \mid (x,y,z) \text{ in } \{(1,2,3), (4,5,6)\}\} \\
&= \text{sum } \{1 \times 2, 4 \times 5\} \\
&= \text{sum } \{2,20\} \\
&= 22
\end{align*}
\]
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) \\
    \mid (img,hdr) in Inputs\} in \\
    let Reslices := \{reslice(wp) \\
    \mid 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

\[
\begin{align*}
v & \rightarrow c \\
\text{or} & \quad \vdots \\
\text{or} & \quad \{ \text{elem} \}
\end{align*}
\text{or}

\begin{align*}
v & \text{copy} \\
\text{or} & \quad \vdots \\
\text{or} & \quad \{ \text{elem} \}
\end{align*}
\]
Example value
Next step: evaluation nodes ("process")

Constants, primitive functions

Variables & temporary bindings

\[
\text{let}_x e_1, \ldots, e_n \\
\]
Pairing

Record building

Field lookup

$\langle \rangle \\
\pi_A$

$A_i \\
A_n$
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
Input values
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
if $e_2$ then test else if $e_1$ then else copy

if $e_2$ then test else if $e_1$ then else copy
OK, take a deep breath!
An example

\[
\text{for}_x \{ \text{head} \rightarrow \text{elem} \downarrow \text{body} \} + \{ \text{elem} \downarrow \text{head} \} \text{elem} \rightarrow x
\]
An example

```
{ 1, elem 2, elem } for_{x} + 1, x
```
An example
An example

```
An example

for x in l:
    x = C
```

![Diagram](image-url)
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
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

- 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)
Graphs are partially "replayable"

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

Formalized in (Acar, Ahmed, Cheney 08)

Stuck!

- 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