Hierarchical models of provenance

Peter Buneman
James Cheney
Egor Kostylev

University of Edinburgh
TaPP, June 15, 2012
Motivation

• Many provenance systems track "flat" provenance

• Some track provenance at multiple granularity levels (in different ways)
  • e.g. ZOOM, Kepler, probably others

• Our goals:
  • Formal, high-level model of "hierarchical" provenance
  • Understand interplay between control/data abstractions and provenance models
OPM lite

- Simplified OPM: bipartite DAGs
  - Process nodes \( P \)
  - Artifact nodes \( A \)
  - Edges \( E \subseteq (P \times A) \cup (A \times P) \)
  - Labels on process, artifacts and edges
def f(x) = x+1
g(x,y) = h(x) + x*y
h(z) = z*z
in g(f(1), 4)
def f(x) = x+1
    g(x,y) = h(x) + x*y
    h(z) = z*z
in g(f(1), 4)

Note: This **discards** a lot of information!
"Hierarchical" OPM

- Augment OPM graph structure with *call tree*
- Tree $T = (V, F)$ with nodes labeled by "higher-level" processes
- Mapping $\Omega : V \rightarrow (P \cup A)$
  - Requires if $(f, g) \in F$ then $\Omega(f) \supseteq \Omega(g)$
    - note reversal!
  - $\Omega(\text{root}) = (P \cup A)$
- Further requirements: $\Omega(f)$ is a contiguous, "sub-OPM" graph.
  - Formalized more carefully in paper.
def f(x) = x+1
  g(x, y) = h(x) + x*y
  h(z) = z*z
in g(f(1), 4)
```python
def f(x) = x + 1
g(x, y) = h(x) + x * y
h(z) = z * z
in g(f(1), 4)
```
def f(x) = x+1
  g(x,y) = h(x) + x*y
  h(z) = z*z
in g(f(1), 4)
Views

- Given a prefix-closed subtree $S$ of $T$
- This induces a *view* of $H$
  - that is, an normal OPM graph
- obtained by "collapsing" the graph structure of calls not in $S$
- This makes sense (only) because of restrictions on call mapping $\Omega$.
  - (details in paper - there are a few subtleties)
def f(x) = x+1
  g(x,y) = h(x) + x*y
  h(z) = z*z
in g(f(1), 4)
def f(x) = x+1
g(x, y) = h(x) + x*y
h(z) = z*z
in g(f(1), 4)
def f(x) = x+1
    g(x, y) = h(x) + x*y
    h(z) = z*z
    in g(f(1), 4)
ProvL: Simple "workflows" (ie functional programs)

- We first consider workflows with function calls but little else...

\[ e ::= c \mid x \mid \odot(\overline{e}) \mid \text{let } x = e_1 \text{ in } e_2 \]

- \( \odot \) denotes an arbitrary primitive function
  - think +, *, -, etc.

- Let-binding allows for sharing
  - hence, expression basically isomorphic to graph...
Adding function calls

- We allow (closed, first-order) function definitions, with calls:

\[ e ::= \ldots | f(\vec{e}) \]

\[ \text{def } f_1(\vec{x}_1) = e_1, \ldots, f_m(\vec{x}_m) = e_m \text{ in } e' \]

- Functions can be defined mutually recursively

- Function calls generate call tree nodes (mapped to appropriate subgraphs).
Adding lists, map

- Finally we consider lists and mapping

\[ e ::= \ldots | \text{map}_f(e) \]

- and allow \texttt{nil}, \texttt{cons} (and maybe others) as built-in functions

- Note that \texttt{map} is second-order - i.e. \( \text{map}_f \) is a function for any \( f \)

- Map nodes link lists to lists, sub-call nodes map elements to elements
Map-incr example

def f(x) = x + 1
in map f([1,2,3])
Adding conditionals

- Next we consider if-then-else conditionals.

\[ e ::= \ldots | \text{if } e_1 \text{ then } e_2 \text{ else } e_3 \]

- The graph marks the conditional and direction taken.
def f(x) = if x = 0
    then []
    else x::f(x-1)

h(z) = z*z

in map_h(f(3))
def f(x) = if x = 0
then []
else x::f(x-1)
h(z) = z*z
in map_h(f(3))
Next steps

- Modeling of granularity in existing WF languages (or D-OPM)
  - expressiveness / query languages?
- Extensions to OPM / W3C PROV for hierarchical process structure?
  - complementing / clarifying work on "collections", "accounts"
- Richer language features
  - first-class higher-order functions? (cf. Perera et al. 2012)
  - meta-programming/provenance of provenance ("eval")
- Efficient implementation (exploit redundancy?)
Related work

- Provenance layering libraries/architecture (Muniswamy-Reddy et al. USENIX 2009)
- Builds on / variant of "graph model of workflow and DB prov" (Acar et al. TaPP 2010)
- ZOOM* User Views (Liu et al. TODS 2011)
  - abstract views based on user preferences
- Kepler (Anand et al. EDBT 2010)
  - data are serialized XML streams, QL supports nesting and process step navigation
- and slicing for program comprehension & provenance (Acar et al. 2012, Perera et al. 2012)
  - language-based approach, does not (directly) address abstraction/granu
Conclusions

- Provenance granularity is an important feature of several models
- There is no common understanding for what it means or how it should work
- Our contribution: basic model of "hierarchical" OPM
- But mostly open questions
Paper/appendix gives operational semantics producing HOPM graphs

\[
\begin{align*}
\frac{(a := \text{Gen}_a(c))}{\gamma, c \downarrow \mathcal{H}(a, -, -), a} & \quad 
\frac{\gamma, \bar{e} \downarrow \mathcal{H}, \bar{a} \quad (a := \text{Gen}_a(\circ(\bar{a}))) \quad (p := \text{Gen}_p(\circ))}{\gamma, (\circ(\bar{e})) \downarrow \bigcup \mathcal{H} \cup \mathcal{H}(a, p, \bar{a}), a} \\
\frac{\gamma, e \downarrow \mathcal{H}, a \quad (a^f = \text{true}) \quad \gamma, e_1 \downarrow \mathcal{H}_1, a_1 \quad (a_n := \text{Gen}_a(a_1^n)) \quad (p_n := \text{Gen}_p(\text{iftrue}))}{\gamma, \text{if } e \text{ then } e_1 \text{ else } e_2 \downarrow \mathcal{H} \cup \mathcal{H}_1 \cup \mathcal{H}(a_n, p_n, (a, a_1)), a_n} \\
\frac{\gamma, e \downarrow \mathcal{H}, a \quad (a^f \neq \text{true}) \quad \gamma, e_2 \downarrow \mathcal{H}_2, a_2 \quad (a_n := \text{Gen}_a(a_2^n)) \quad (p_n := \text{Gen}_p(\text{iffalse}))}{\gamma, \text{if } e \text{ then } e_1 \text{ else } e_2 \downarrow \mathcal{H} \cup \mathcal{H}_2 \cup \mathcal{H}(a_n, p_n, (a, a_2)), a_n} \\
\frac{\gamma, \bar{e} \downarrow \mathcal{H}, \bar{a} \quad (\gamma(f) = f(\bar{x}).e) \quad \gamma\{\bar{x}/\bar{a}\}, e \downarrow \mathcal{H}, a}{\gamma, f(\bar{e}) \downarrow \bigcup \mathcal{H} \cup \mathcal{H}_f(a, \bar{H}, \bar{a}), a} \\
\frac{\gamma, e \downarrow \mathcal{H}, a \quad (a^f = [c_1, \ldots, c_n]) \quad \gamma, f(c_1) \downarrow \mathcal{H}_1, a_1 \quad \ldots \quad \gamma, f(c_n) \downarrow \mathcal{H}_n, a_n \quad (a' := \text{Gen}_a([a_1^n, \ldots, a_n^n]))}{\gamma, \text{map}_f(e) \downarrow \mathcal{H} \cup \mathcal{H}_f^{\text{map}}(a, [\mathcal{H}_1, \ldots, \mathcal{H}_n], a'), a'} \\
\frac{\gamma, \text{def } \tilde{f}(\bar{x}) = \bar{e} \text{ in } e \downarrow \mathcal{H}_{\text{main}}(\mathcal{H})}{\text{def } \tilde{f}(\bar{x}) = \bar{e}}
\end{align*}
\]