On the traceability in a graph roundtrip transformation system GRoundTram†

Soichiro Hidaka
National Institute of Informatics

Workshop on Bidirectional Transformation (BT2010)
Mar. 15th, 2010

†Joint work with
Zhenjiang Hu, Kazuhiro Inaba, Hiroyuki Kato (National Institute of Informatics)
Kazutaka Matsuda (University of Tokyo)
Keisuke Nakano (University of Electro-Communications)
Isao Sasano (Shibaura Institute of Technology)
Since BT-ABC-3 (Nov. 2009)

• In the BT-ABC-3
  – support bidir. interpretation of much-more user-friendly surface syntax
  – lots of examples added
  – supports (limited form of) insertion
  – graph editor is provided
  – released complete source code in OCaml
  – 17000 lines of codes including comments

• At the time of BT-ABC-4
  – More clear formulation of trace information
  – Deletion support (not implemented yet)
  – Enhanced handling of insertions
  – Graph contraction support
• **Graph Roundtrip Transformation for Models**
  – to support systematic development of bidirectional model transformation

• ground tram
  – bidirectional ground transportation system (tram usually runs in both directions)

• "Ground" ... we would like to provide "basis"
  – user-level model representation is internally represented by more general graph
  – base graph manipulation system (graph algebra) to be
    • **powerful** enough to express typical model transformation
    • **rigid** enough to reason about properties such as well-behavedness
Model Driven Software Development

• We want to reflect modification at the downstream to the upstream
Bidirectional Model Transformation Example: Class2RDB

Class
- name = "Person"
- is_persistent = true
  - attrs
    - Attribute
      - name = "name"
        - is_primary = true
      - PrimitiveDataType
        - name = "String"
  -Association
    - name = "address"
      - src_of
        - Class
          - name = "Address"
          - is_persistent = false
            - attrs
              - Attribute
                - name = "addr"
                  - is_primary = true
              - PrimitiveDataType
                - name = "String"
    - dest
      - Class
        - name = "Phone"
        - is_persistent = true
          - attrs
            - Attribute
              - name = "number"
                - is_primary = true
            - PrimitiveDataType
              - name = "Integer"

Table
- name = "Person"
  - Column
    - name = "name"
      - type = "String"
      - pkey
        - cols
          - ref
            - Table
              - name = "Phone"
                - Column
                  - name = "number"
                    - type = "Integer"
                    - Fkey
                      - cols
                        - ref
                          - Table
                            - name = "Person"
                              - Column
                                - name = "name"
                                  - type = "String"

S. Hidaka  Traceability in GRoundTram
Our Target Bidirectional Transformation

\[ s \xrightarrow{\text{forward Trans.}} t \]

\[ s' \xleftarrow{\text{backward Trans.}} t' \]

- Forward Transformation
- Backward Transformation
- Modify
Why UnQL/UnCAL?

- Functional (Compositional)
- Has clear formal semantics
- Has nice user-level syntax
Basic idea of compositional approach in BT

complecation: variable reference
Encoding in Graphs

- Edge label: **primitive data and tag**
- Set of outgoing edges: **collections**
- Edges directed to other node: **references** to other components

Issue: coping with bisimulation
A compositional Framework for Bidirectional Model Transformation

MQL
Two methods for typechecking

Model Transformation in UnQL+[1] (Compositional and Functional)

desugaring (incl. editing primitives) to core language

UnCAL graph algebra (Graph Construction and Structural Recursion)

Validation

Bidirectional Interpreter
- Bidirectionalization
- Fusion Optimization

Validation

source model

target model

[1] SAC2009

[Overview]
ICSE2009
NIER Tr.
Project site

- www.biglab.org

The BiG Project

A Grand Challenge Project on Bidirectional Graph-Model Transformation at NICT in Japan

Top Demo

BiG project bidirectional UnQL/UnCAL

You can issue UnQL or UnCAL query on source DB by selecting the input and the query from the list boxes below.

Select source DB

S. Hidaka Traceability in GRoundTram
Agenda

• Traceability
  – Enriched forward transformation
  – Tracing source node from target node
  – Tracing source edges from target edges
• Tracing in deletion handling
• Tracing in insertion handling
• Internal representation and view edited by end-users
• Conclusion and future work
UnCAL graph model

- Cyclic
- Edge-labeled
  - All informations are stored in edge labels
  - Node ids have no particular meanings
- Directed
- Unordered (branches)
- Multi-rooted (has entry points as input nodes)
  - Indicated by input markers
- Has (optional) output nodes
  - Indicated by output markers
- Unranked
  - Node has arbitrary number of edges
- Has ε edges

**Property**: Every UnCAL graph can be normalized up to isomorphism.
Source graph example: Customer-centric order information

- **Customer2Order** (Pastor et al. 2007)
Transformation in UnQL

```sql
select
  {order: {date: $date, no: $no,
    customer_name: $name, addr: $a}}

where
  {customer.order: $o} in $db,
  {order_of:$c, date:$date, no:$no} in $o,
  {add:$a, name:$name} in $c,
  {code:$code, info:$info, type:shipping} in $a
```
Target graph example: Order-centric graph

```sql
select
    {order: {date: $date, no: $no,
              customer_name: $name, addr: $a}}
where
    {customer.order: $o} in $db,
    {order_of:$c, date:$date, no:$no} in $o,
    {add:$a, name:$name} in $c,
    {code:$code, info:$info, type:shipping} in $a
```

S. Hidaka   Traceability in GRoundTram
From UnQL to UnCAL via structural recursion

- Extract every subgraph under edges labeled b

\[
\begin{align*}
\text{select } G & \quad \text{sfun } f(\{b:G\}) = G \\
\text{where } \{*_b:G\} \text{ in } db & \quad \text{rec}(\lambda (\{l,G\}). \text{if } l = b \text{ then } G \\
\text{in } f(db) & \quad \text{else } \{ε:&\}(db)
\end{align*}
\]

**Property:** Every UnQL query can be translated into UnCAL query

S. Hidaka  Traceability in GRoundTram
UnCAL Syntax (Subset)

Buneman et al., VLDBJ2000

\[
T ::= \{
\} \quad (* \text{one-node graph} *)
| \{L : T\} \quad (* \text{graph with a single edge from root} *)
| T \cup T \quad (* \text{union of two graphs} *)
| \&x := T \quad (* \text{label the root node with input marker } x *)
| \&y \quad (* \text{one-node graph with output marker } y *)
| () \quad (* \text{empty graph} *)
| T \oplus T \quad (* \text{disjoint union} *)
| T \circ T \quad (* \text{append of two graphs} *)
| \text{cycle}(T) \quad (* \text{graph with cycles} *)
| G \quad (* \text{variable reference} *)
| \text{if } L = L \text{ then } T \text{ else } T \quad (* \text{conditional} *)
| \text{rec}(\lambda(l,G).T)(T) \quad (* \text{application of structural recursion} *)
\]

\[
L ::= a \quad (* \text{edge label} *)
| l \quad (* \text{label variable reference} *)
\]
Bulk Semantics of UnCAL
Structural recursion (example)

\[ \text{sfun}\ a2d\_xc(\{l:\!g\}) = \begin{align*}
&\text{if } l = a \text{ then } \{d : (a2d\_xc(\!g))\} \\
&\text{else if } l = c \text{ then } a2d\_xc(\!g) \\
&\text{else } \{l : (a2d\_xc(\!g))\} \\
\end{align*} \]

in a2d\_xc(\!db)

Transformation to core algebra (desugaring)

\[ \&z_1 @ \text{rec}( \lambda \{l,\!g\} ). \begin{align*}
&\text{if } l = a \text{ then } \&z_1 := \{d : \&z_1\} \\
&\text{else if } l = c \text{ then } \&z_1 := \{\varepsilon : \&z_1\} \\
&\text{else } \&z_1 := \{l : \&z_1\}(\!db) \\
\end{align*} \]
Bulk Semantics of UnCAL
Structural recursion (example)

S. Hidaka  Traceability in GRowndTram
Union(∪)  
The Role of $\varepsilon$-edges

- Remember (store) the separation points

How do we decide?

Instead
The Role of $\varepsilon$-edges (cont.)

- Given modified tree $t'$
  - Even in the presence of overlap
- Compute $\text{reachable}(g, v)$ to grab reachable parts of the graph $g$ from node $v$
- And apply backward transformation recursively
Union(∪) Multiple Input Nodes

- Take reachable parts from the original input nodes
- Had we removed ε edges, decomposition would not have been possible
TraceID ::= DatabaseID
   | C Pos Marker
   | RN Pos TraceID Marker
   | RE Pos TraceID Edge

\[
\begin{align*}
\text{tr}(v) &= v \quad (v \in \text{DatabaseID}) \\
\text{tr}(RN \_ v \_) &= \text{tr}(v) \\
\text{tr}(RE \_ v \_) &= \text{tr}(v)
\end{align*}
\]

\[
\begin{align*}
\text{corr} \ (\langle v, l, u \rangle) &= \langle v, l, u \rangle \quad (v, u \in \text{DatabaseID}) \\
\text{corr} \ ((\text{RE} \_ e', l, \text{RE} \_ e')) &= \text{corr}(e')
\end{align*}
\]
Bulk semantics and bidirectionalization of structural recursion

rec(\lambda (\{$/,$G\}). if $l = b$ then $G$ else {\varepsilon:\&})($db$)

Internal representation retaining trace information

Also a unit of insertions

Graphs produced by the body of rec for each input edge
Epsilon edge removal: problematic case

- When removing $\varepsilon$ edges, the source and the target of the edge can’t always be glued together.

- Node 1 should not reach node 5!
General condition in which the gluing fails

- If the target of the \( \varepsilon \) edge has another incoming edge, and the source of the \( \varepsilon \) edge has another outgoing edge, then the gluing fails.

![Diagram showing the gluing condition](image)
Epsilon edge removal: correct method in UnQL paper

- Roughly, outgoing edges of the target of the $\epsilon$ edges are copied.
  
- If the target has no outgoing edges, then nothing should be done.
  
- Node 4 and edge $b$ is no longer reachable and can be removed as well.
\(\varepsilon\) edge removal and transitive closure

- **Example**

- **General form**
Can these problematic cases be generated from UnQL(+)?

- Yes, but only from limited regular path pattern, and I think it is rare.

- For ordinary combination of constructors (cycle, append and union), no problem.

```sql
select $G
where {((b.(a|x))|(c.(a|a.y)):G} in $db
```

```
&s_1 := a U $G, &s_2 := a U $G, ...
```

S. Hidaka  Traceability in GRoundTram
Simple and concrete example of problematic case

let $G = \{c\}$

in $(\&x1 := \{a\} \cup G, \&x2 := \{b\} \cup G)$

(*) UnCAL Source *)

Naïve gluing of

Before $\varepsilon$ removal

input node $\&x2$ should not reach edge “a”!

Correct solution
Current implementation

• Formal solution generally introduces unreachable parts
  – It will hinder current clear separation of $\varepsilon$ removal and unreachable part removal

• Process split in two phases
  – 1$^{st}$ stage tries gluing for only safe cases
  – 2$^{nd}$ stage does the rest

• If ambiguity found during the process
  – Entire epsilon removal is given up
Conclusion

• Enhancement of forward semantics for traceability
• Deletion handling using edge tracing
• Insertion handling using node tracing
• Correspondence between internal representation and view presented by end users

• Future work
  – More user-friendly editing support
  – More clear understanding of relationship between (constant) complement approach