Practical Aspects of
Bidirectional Graph Transformations

Zhenjiang Hu
National Institute of Informatics (NII)
January 21, 2013

Joint Work with the BiG Team Members
BiG: An NII Grand Challenge Project (2008-Now)

Aim at **Language-based Foundation of Bidirectional Graph Transformation for Evolutionary Software Development**

Zhenjiang Hu
Prof. (NII), Project Leader
BX Programming

Hiroyuki Kato
Assist. Prof. (NII)
BX Optimization

Kazuyuki Asada
Post-doc (NII)
BX Semantics

Soichiro Hidaka
Assist. Prof. (NII)
Bidirectionalization

Kazuhiro Inaba
Researcher (Google)
BX Verification

Keisuke Nakano
Assoc. Prof. (Univ. of Electro Comm.)
BX Analysis

Isao Sasano
Assist. Prof. (Shibaura Univ.)
BX Application

Collaborators (U. Tokyo, OU, PKU, SJTU, Oxford U, Minho U, INRIA)

PEPM 2013
Zhenjiang Hu, Shin-Cheng Mu, Masato Takeichi,

A Programmable Editor for Developing Structured Documents based on Bidirectional Transformations,

Bidirectional Transformation
Bidirectional Transformation (BX)

[Nate Foster, et al: POPL 2005]
Roundtrip Properties

Get-Put:
\[ \text{put } s \ (\text{get } s) = s \]

Put-Get:
\[ \text{get } (\text{put } s \ t) = t \]
Pervasive Bidirectional Transformation

Bidirectional Computation

BiX: A Bidirectional Tree Transformation Language

Bidirectional Model Transformation: A Compositional Approach

Conclusion

Definition

Basic Properties

Direct Applications

View Updating

Reflect changes on the view to the original relational database.

Ref:
Many studies in the database community


Zhenjiang Hu

Data Synchronization (2006)

Replicated Data Synchronization

Synchronization of data in different formats.

Ref:
The Harmony project in University of Pennsylvania

[Pierce et al.: POPL'05, PODS'06, POPL'08]

Zhenjiang Hu

Model Code Coevolution (2012)

Model Code Coevolution (2012)

Model

Code

View Updating (80')

DB

View

PEPM 2013
Bidirectional Transformation Languages

- **We need languages to support development of software with bidirectional computation**

  - **Lens** (for data sync) Univ. of Pennsylvania, ...
    - [POPL’05, PODS’06, ICFP’08, ICFP’10, POPL’11,’12]

  - **X/Inv/BiX** (for doc construction) Univ. of Tokyo / NII
    - [PEPM’04, MPC’04, ICFP’07, ESOP’10, ICFP’10]
Bidirectional Graph Transformations
BX on Graphs is Wanted!

Models: Graphs
Bidirectional Model Transformation: Bidirectional Graph Transformation
Can We Design a Language for BiG?

Bi-directional Graph Transformation

PEPM 2013
BiG is a Challenge!

Graphs have node sharing and cycles

- How to deal with termination of graph transformation?
- How to deal with equality of two graphs?
- How to correctly reflect changes on the view to the source?
TGG Framework [Schurr et al.]

- TGG (triple graph grammar) + Rule-based
  - MOFLON
  - QVT: Query/View/Transformation

Issues:
- compositional
- well-behavedness
- scalability
Domain Specific Framework

• Bidirectionalization of a subset of ATL (ASE’07)

• Beanbag: model synchronization (FSE’09)
Towards a General Solution …

• How to deal with termination of graph transformation?
  ➔ Structural Recursion (fold on graphs)

• How to deal with equality of two graphs?
  ➔ Bisimulation (graphs as regular trees)

• How to correctly reflect changes on the view to the source?
  ➔ Traceability based on Bulk Semantics
GRoundTram: A General Functional Framework

- It is **compositional** *(functional)*
  - Based on the existing graph query language unQL
- It is **well-behaved**
  - Built upon bidirectional UnCAL: a graph algebra with clear bidirectional semantics
- It is an **integrated development environment**
  - Graph editor, graph validation, graph transformation checking, visualizations of bidirectional behavior
GRoundTram: Architecture

- Source Model (DOT/UnCAL)
- Source Schema (KM3)
- Transformation (UnQL+)
- Target Schema (KM3)

Model Validation

Model Transformation Validation

Verified Transformation

- Source Model (DOT)
- Graph Update
- Updated Source Model (DOT)

Forward Transformation

Bidirectional Transformation

- Target Model (DOT)
- Graph Update
- Updated Target Model (DOT)

PEPM 2013

SAC’09

PPDP’11

ICFP’10, LOPSTR’11

User Input
Bidirectionalization

BiG in UnQL+ (forward)

Graph Algebra (Structural Recursion)

Bidirectional Semantics

Source Graph ↔ Target Graph

(ICFP 2010, LOPSTR 2011)
Bidirectionalization

BiG in UnQL+ (forward)

Graph Algebra (Structural Recursion)

Bidirectional Semantics

Source Graph

Target Graph

(ICFP 2010, LOPSTR 2011, ICSE 2012)

PEPM 2013
Graph Model

- **Rooted Edge-labeled Graph**

\[
G = (V, E, I, O)
\]

where

\[
V = \{1, 2, 3, 4, 5, 6\}
\]

\[
E = \{(1, a, 2), (1, b, 3), (1, c, 4),
(2, a, 5), (3, a, 5), (5, d, 6)\}
\]

\[
I = \{(\&, 1)\}
\]

\[
O = \{\}
\]

\[
G_{\text{root}} = \{a : \{a : G_5\}, b : \{a : G_5\}, c : G_4\}
\]

\[
G_5 = \{d : \{\}\}
\]

\[
G_4 = \{c : G_4\}
\]
Structuring Graphs as Regular Trees

• Graph Equivalence based on Bisimulation
Example: A Customer-Order Graph
Bidirectionalization

BiG in UnQL+ (forward)

Graph Algebra (Structural Recursion)

Bidirectional Semantics

Source Graph  Target Graph

(ICFP 2010, LOPSTR 2011)
UnQL+

- An extension of UnQL, a functional graph query language [Buneman et. al. VLDB’00].
  - select ... where ...
  - replace ... by ... where ... [SAC’09]

```python
select {order: {date: $date,
    no: $no,
    name: $name,
    addr: $a}}
where {customer.order: $o} in $customer_db,
{order_of: $c, date: $date, no: $no} in $o,
{add: $a, name: $name} in $c,
{type: shipping} in $a
```
Bidirectionalization

BiG in UnQL+
(forward)

Graph Algebra
(Structural Recursion)

Bidirectional Semantics

Source Graph

Target Graph

(ICFP 2010, LOPSTR 2011)
Graph Constructors

They can be used to construct arbitrary graphs, non-uniquely.
Structural Recursion: Folds on Graphs

[Buneman et. al. VLDB’00]

\[
\begin{align*}
    f \{ \} & = \{ \} \\
    f \{ l : g \} & = E[l,g] \ @ \ f \ g \\
    f \ (g1 \cup g2) & = f \ g1 \cup f \ g2
\end{align*}
\]

Or simply written as

\[
Sfun \ f \{ l : g \} = E[l,g] \ @ \ f \ g
\]

\[
f = \text{rec}(\lambda (l,g). E[l,g])
\]
Structural Recursion: Example

\[
\text{sfun } a2d\_xc \ (\{l : g\}) = \\
\text{if } l = a \text{ then } \{d : a2d\_xc(g)\} \\
\text{else if } l = c \text{ then } a2d\_xc(g) \\
\text{else } \{l : a2d\_xc(g)\}
\]
Bidirectionalizing Structural Recursion

```
sfun a2d_xc (\l : g) =
  if \l = a then \{d : a2d_xc(g)\}
  else if \l = c then a2d_xc(g)
  else \{l : a2d_xc(g)\}
```
Bidirectionalizing Structural Recursion

\[
sfun \ a2d_{xc} (\{l : g\}) = \begin{cases} 
    \{d : a2d_{xc}(g)\} & \text{if } l = a \\
    \text{if } l = c \text{ then } a2d_{xc}(g) & \text{else if } \text{else } \{l : a2d_{xc}(g)\}
\end{cases}
\]
Bidirectionalizing Structural Recursion

\[
sfun \ a2d\_xc \ (\{l : g\}) = \begin{cases} 
  d : a2d\_xc(g) & \text{if } l = a \\
  \text{else if } l = c & \text{then } a2d\_xc(g) \\
  \text{else } \{l : a2d\_xc(g)\}
\end{cases}
\]
Optimization on Structural Recursion

- **Fusion Law**
  \[ \text{rec}(e_1) (\text{rec}(e_2)(G)) \rightarrow \text{rec}(\ldots)(G) \]

- **Tupling Law**
  \[ (\text{rec}(e_1)(G), \text{rec}(e_2)(G)) \rightarrow \text{rec}(\ldots)(G) \]

- **Decomposition Law**
  \[ \text{rec}(e) \rightarrow \text{rec}_s(e_1) . \text{rec}_j \]
Applications
GRoundTram: Snapshot

Graph Roundtrip Transformation of Models
http://www.biglab.org/
GRoundTram as a Backend BX Engine

Automatic Feature Model Fixing  
Towards automatic Bidirectionalization of ATL  
Model-Code Co-evolution

(MODELS’10)  (ICMT’11)  (ICSE’12)
An Application: blinkit (ICSE 2012)

- A tool for supporting invariant traceability in model-driven development

(Joint work with Yijun Yu at Open Univ.)
An Application: blinkit (ICSE 2012)

- A tool for supporting invariant traceability in model-driven development

(Joint work with Yijun Yu at Open Univ.)
Experiments

Are there realistic cases in an open-source MDD software development project where invariant traceability links can be synchronized using blinkit?

EMF/GMF
- 6 years CVS repository
- 28,070 file revisions
- 1,185 revision pairs
- 178 @model changes
- 146,415 @generated NOT methods
- 9.61 @generated INV cases per model revision

http://sead1.open.ac.uk/linkit/
Practical Aspects of BiG

Predictability
Expressiveness
Scalability
Dependability
Existing Approaches to Bidirectional Programming

- Domain Specific Bidirectional Languages
- Automatic Bidirectionization of ATL, XQuery, UnQL

Assumption:
Each get has one fixed put.
Not a Practical Assumption!

Since get is generally non-injective, many suitable puts correspond to one get, each being useful in different context.
Ambiguity in Backward Transformation

Select *
From EMP
Where Location = "Tokyo"

1. Delete Tanaka?
2. Move Tanaka from Tokyo to Kyoto?

Delete Tanaka
### Ambiguity in Backward Transformation

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Location</th>
<th>BBTeam</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tanaka</td>
<td>Tokyo</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Kato</td>
<td>Tohoku</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Sato</td>
<td>Tokyo</td>
<td>No</td>
</tr>
</tbody>
</table>

1. **Delete Tanaka?**
2. **Move Tanaka out from BBTeam?**

Select *
From EMP
Where BBTeam = Yes

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Location</th>
<th>BBTeam</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Tanaka</td>
<td>Tokyo</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>Kato</td>
<td>Tohoku</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Delete Tanaka**
A Solution: Our Ongoing Work …

- Do backward bidirectionalization

\begin{itemize}
  \item \textit{get” (forward transformation)}
  \begin{itemize}
    \item \textit{put” (backward transformation)}
  \end{itemize}
  \item \textit{get” (forward transformation)}
  \begin{itemize}
    \item \textit{put” (backward transformation)}
  \end{itemize}
\end{itemize}
Injectivity: get and (put s)

• Injective get is too limited for BX

\[ \text{add (x, y)} = x+y \]

• Injective (put s) is powerful and necessary for BX

\[ \text{putAdd (x,y) v} = (v-y, y) \]

Essence of BX Programming

BX Programming = Writing injective (put s) + Deriving get
Injectivity of “put s”: Necessity

**Lemma**: If there exists a get satisfying PutGet, put s must be injective.

**Proof**:

\[
\text{put } s \ v = \text{put } s \ v' \Rightarrow \{ \text{Leibniz} \} \\
\text{get } (\text{put } s \ v) = \text{get } (\text{put } s \ v') \Rightarrow \{ \text{PutGet} \} \\
\ v = \ v'
\]
Derived “get”: Uniqueness

**Lemma**: Given a put function, there exists at most one get function such that GetPut and PutGet hold.

**Proof**: Suppose we have two get functions, say get and get’.

\[
\begin{align*}
\text{get } s \\
\Rightarrow \{ \text{GetPut} \} \\
\text{get } (\text{put } s \ (\text{get’ } s)) \\
\Rightarrow \{ \text{PutGet} \} \\
\text{get’ } s
\end{align*}
\]
Well-behaved “put”

**Definition:** A “put” function is said to be well-behaved, if there exists a (unique) “get” such that GetPut and PutGet hold.

**Question:** Are the following put function behave well?
- put1 \( s \ v = s \)
- put2 \( s \ v = s ++ v \)
- put3 \( (a : s) \ v = a : v \)

Difficult to check because we do not have “get” yet ...
Well-behaved “put”

Theorem:
Put is well-behaved, iff
1. Put $s$ is injective
2. Put is surjective: For any $s$, there exist $s'$, $v$, such that $s = \text{put } s' v$
3. Put is stable: $\text{put } (\text{put } s v) v = \text{put } s v$

Question: Are the following put function behave well?

- $\text{put1 } s v = s$
- $\text{put2 } s v = s ++ v$
- $\text{put3 } (a : s) v = a : v$
“Put” Programming in Curry

```haskell
type Put s v = s -> v -> s

get :: Put s v -> s -> v
get put s | s == put s v = v

where v free

uHead :: Put [a] a
uHead [] v = [v]
uHead (a:s) v = v:s

uPair :: Put (Int, Int) Int
uPair (x,y) v = (v, v-x-y)
```

Derive “get” by “needed narrowing”

Write “put” freely, and check it with “quickcheck”

Sebastian Fischer, Zhenjiang Hu and Hugo Pacheco
“Putback” is the Essence of Bidirectional Programming,
GRACE-TR 2012-08, GRACE Center, National Institute of Informatics, December 2012, 36 pages.
Conclusion

• A linguistic approach to BiG
  - Representing graphs by regular trees
  - Manipulating graphs with structural recursion
  - Bidirectionalizing graph transformations using Bulk semantics

• A BiG system GRoundTram
  - An integrated environment
  - Applications for model synchronization and model-code coevolution (evolutionary software development)

• Practical challenges
  - Predictability: put-based bx programming
  - Expressiveness: handling graphs with ordered branches
BiG Project Web Site

For more information, please visit the project page which contains all published papers as well as the source codes of the GRroundTram system.

http://www.biglab.org

Join us?