Bidirectional Transformation and Its Application to Dependable Document Construction

Zhenjiang Hu and Masato Takeichi

Department of Mathematical Informatics
The University of Tokyo

December 7, 2007

(Joint Work with Y. Hayashi, D. Liu, S.C. Mu and K. Nakano)
Bidirectional Transformation and Its Application to Dependable Document Construction

Zhenjiang Hu and Masato Takeichi

Department of Mathematical Informatics
The University of Tokyo

December 7, 2007

(Joint Work with Y. Hayashi, D. Liu, S.C. Mu and K. Nakano)
Bidirectional Transformation and Its Application to Dependable Software Construction

Zhenjiang Hu and Masato Takeichi

Department of Mathematical Informatics
The University of Tokyo

December 7, 2007

(Joint Work with Y. Hayashi, D. Liu, S.C. Mu and K. Nakano)
Bidirectional Transformation and Its Application to Dependable Document Construction

Zhenjiang Hu and Masato Takeichi

Department of Mathematical Informatics
The University of Tokyo

December 7, 2007

(Joint Work with Y. Hayashi, D. Liu, S.C. Mu and K. Nakano)
Outline

1 Bidirectional Transformation
   - Bidirectional Transformation
   - Basic Bidirectional Properties
   - Two Direct Applications

2 Bidirectional Transformation in Document Construction

3 BiX: A Bidirectional Transformation Language

4 From Document Engineering to Software Engineering
An idea originated from the **view-updating technique** in the DB community.
Bidirectional Transformation

An idea originated from the view-updating technique in the DB community.

![Diagram with src and tgt triangles]
An idea originated from the *view-updating technique* in the DB community.
An idea originated from the **view-updating technique** in the DB community.
An idea originated from the **view-updating technique** in the DB community.

![Diagram showing bidirectional transformation](image)
An idea originated from the *view-updating technique* in the DB community.
An idea originated from the view-updating technique in the DB community.

It consists of a pair of transformations: forward and backward.
Stability

No change on the target implies no change on the source.

\[
\text{put}(\text{get}(s), s) = s
\]
Reflectivity

Permitted changes on the target should be reflected to the source.

\[
\text{get}(\text{put}(t', s)) = t'
\]
Ref: many studies in the database community
[Bancilhon&Spyratos:81, Dayal&Bernstein, Gottlob et. al.:88]
Replicated Data Synchronization

Synchronization of data in different formats.

Ref: the Homony project in Univ. of Pennsylvania
[Pierce et. al: POPL’05, PODS’06, POPL’08].
Outline

1. Bidirectional Transformation

2. Bidirectional Transformation in Document Construction
   - Document Engineering
   - The PSD Project
   - Two Core Techniques

3. BiX: A Bidirectional Transformation Language

4. From Document Engineering to Software Engineering
Document engineering is concerned with principles, tools and processes that improve our ability to create, manage, and maintain documents.
Document engineering is concerned with principles, tools and processes that improve our ability to create, manage, and maintain documents.

Documents:

- Document Source
- Document View
- View Generator
Document engineering is concerned with principles, tools and processes that improve our ability to create, manage, and maintain documents.

Documents:
- Document Source: XML source
- Document View: HTML
- View Generator: XSLT
**Document engineering** is concerned with principles, tools and processes that improve our ability to create, manage, and maintain documents.

**Documents:**
- **Document Source**  XML source
- **Document View**   HTML
- **View Generator**  XSLT

Documents are of much simpler structures than softwares.
The PSD Project

A bidirectional transformation framework for supporting formal creation and maintenance of documents.

Motivation: Applying program transformation techniques to document transformation.
**Document Creation**

**XEditor**: Document Construction $\equiv$ View Editing

$$\text{XML} + \text{Transformation in X/Inv} \iff \text{Editing on view}$$

*Ref*: [MPC’04, PEPM’04, HOSC:07].
Vu-X: update web pages on browsers in the WYSIWYG manner.

Ref: [JSSST'06, JSSST'07, PEPM'07, APWeb'07].
Vu-X: update web pages on browsers in the WYSIWYG manner.

http://www.psdlab.org/vux/
Bidirectional Languages

- Inv: an injective language for revertible computation
  [MPC’04, APLAS’04]
Bidirectional Languages

- **Inv**: an injective language for revertible computation  
  [MPC'04, APLAS'04]

- **X**: a domain-specific bidirectional language for interactive document construction (with the *dup* primitive)  
  [ACM PEPM'04, HOCS:07]
Bidirectional Languages

- **Inv**: an injective language for revertible computation
  
  [MPC’04, APLAS’04]

- **X**: a domain-specific bidirectional language for interactive document construction (with the `dup` primitive)
  
  [ACM PEPM’04, HOCS:07]

- **BiX**: a bidirectional transformation language for general XML processing (with function definitions and local bindings)
  
  [ACM PEPM’07]
Bidirectional Languages

- **Inv**: an injective language for revertible computation
  
  [MPC'04, APLAS'04]

- **X**: a domain-specific bidirectional language for interactive
document construction (with the `dup` primitive)

  [ACM PEPM'04, HOCS:07]

- **BiX**: a bidirectional transformation language for general XML
  processing (with function definitions and local bindings)

  [ACM PEPM'07]

- **ArchX**: a domain-specific bidirectional transformation
  language with regular patterns
Bidirectional Languages

- **Inv**: an injective language for revertible computation  
  [MPC’04, APLAS’04]

- **X**: a domain-specific bidirectional language for interactive document construction (with the *dup* primitive)  
  [ACM PEPM’04, HOCS:07]

- **BiX**: a bidirectional transformation language for general XML processing (with function definitions and local bindings)  
  [ACM PEPM’07]

- **ArchX**: a domain-specific bidirectional transformation language with regular patterns

- **Bi-Haskell**: a general bidirectional functional language
Bidirectional Languages

- **Inv**: an injective language for revertible computation  
  [MPC’04, APLAS’04]

- **X**: a domain-specific bidirectional language for interactive document construction (with the `dup` primitive)  
  [ACM PEPM’04, HOCS:07]

- **BiX**: a bidirectional transformation language for general XML processing (with function definitions and local bindings)  
  [ACM PEPM’07]

- **ArchX**: a domain-specific bidirectional transformation language with regular patterns

- **Bi-Haskell**: a general bidirectional functional language

Write forward transformation and get backward transformation for free!
Bidirectionalization of \textit{XSLT} [JSSST-CS:06]
Bidirectionalization

- Bidirectionalization of XSLT [JSSST-CS:06]
- Bidirectional Interpretation of XQuery [ACM PEPM'07]
Bidirectionalization

- Bidirectionalization of \textit{XSLT} [JSSST-CS:06]
- Bidirectional Interpretation of \textit{XQuery} [ACM PEPM'07]
- Bidirectional Interpretation of \textit{ATL} [ACM/IEEE ASE'07]
Bidirectionalization

- Bidirectionalization of \textit{XSLT} \cite{JSSST-CS:06}
- Bidirectional Interpretation of \textit{XQuery} \cite{ACM PEPM'07}
- Bidirectional Interpretation of \textit{ATL} \cite{ACM/IEEE ASE'07}
- Bidirectionalization of a small \textit{general functional language} \cite{ACM ICFP'07}
Outline

1. Bidirectional Transformation
2. Bidirectional Transformation in Document Construction
3. BiX: A Bidirectional Transformation Language
4. From Document Engineering to Software Engineering
A Combinator-based language
A **Combinator-based** language

- **Primitive bidirectional transformations** for tree manipulation, which is extensible.
A Combinator-based language

- Primitive bidirectional transformations for tree manipulation, which is extensible.
- Combinators for composing smaller bidirectional transformations
• A **Combinator-based** language
  
  • **Primitive bidirectional transformations** for tree manipulation, which is extensible.
  
  • **Combinators** for composing smaller bidirectional transformations

We only need to prepare a pair of transformations for each primitive bidirectional transformation, other backward transformations are obtained for free.
Primitive bi-directional transformations for tree manipulation:

- for tree destruction, e.g.,
  - $xid$
  - $xleftchild$

- for tree construction, e.g.,
  - $xconst\ t$
  - $xdup$
  - $xnewroot\ n$
Identity Transformation
Identity Transformation
Identity Transformation
Identity Transformation

\[ s \xrightarrow{\text{get}} s \xleftarrow{\text{put}} t' \xrightarrow{\text{mod}} t' \]
Identity Transformation

\[
\text{get } s = s \\
\text{put } t' s = t'
\]
Select the leftmost child of the root
Select the leftmost child of the root
Select the leftmost child of the root

$$s_1 \quad \ldots \quad s_n$$

$$s'_1$$

mod
Select the leftmost child of the root
**xleftchild**

- Select the leftmost child of the root

\[
\begin{align*}
\text{get } (\text{Node } n \ [s_1, \ldots, s_n]) & = s_1 \\
\text{put } s'_1 \ (\text{Node } n \ [s_1, \ldots, s_n]) & = \text{Node } n \ [s'_1, \ldots, s_n]
\end{align*}
\]
Constant Transformation:
Constant Transformation:

\[ x_{\text{const}} \quad t \]
**Constant Transformation:**

\[ \text{xconst } t \]

\[ \text{s} \rightarrow \text{t} \]

\[ \text{mod} \]

\[ \text{t'} \]


- **Constant Transformation:** disallow any modification on the target.

---

![Diagram](image)
Constant Transformation: disallow any modification on the target.

\[
\begin{align*}
\text{get } s & = t \\
\text{put } t' s & = s
\end{align*}
\]
Replicate data:
Replicate data:
Replicate data:
Replicate data:
Replicate data: allow data dependency in the target

get \( s \) = \( Node - [s, s] \)
put (\( Node - [s_1, s_2] \)) \( s \) = if \( s = s_1 \) then \( s_2 \) else \( s_2 \)

Assume we have only one modification.
Combinators

Used to construct **bigger** bidirectional transformations by composing **smaller** bidirectional transformations.

- Sequential composition: $x_1 ; x_2$
- Parallel composition: $x_1 || x_2$
- Conditional: $x\text{if } p \ x_1 \ x_2$
- Map to each: $x\text{map } x$
Sequential Composition

\[
\begin{align*}
\text{get}(x_1; x_2) \ s & = \ \text{get}_{x_2}(\text{get}_{x_1} \ s) \\
\text{put}(x_1; x_2) \ t' \ s & = \ \text{put}_{x_1}(\text{put}_{x_2} \ t' (\text{get}_{x_1} \ s)) \ s
\end{align*}
\]
Parallel Composition

\[ s_1 \xrightarrow{x_1} t_1 \]

\[ s_2 \xrightarrow{x_2} t_2 \]

\[ s_1 \parallel s_2 \xrightarrow{x_1 \parallel x_2} t_1 \parallel t_2 \]
Map-to-each

\[ s_i \xrightarrow{x} t_i \]

\[ s_1 \ldots s_n \xmap x t_1 \ldots t_n \]
Conditional

If \( p \) holds, then

\[
\begin{array}{c}
\text{\( s \)} \\
\text{\( \rightarrow \)} \\
\text{\( t \)}
\end{array}
\]

\[
\begin{array}{c}
\text{\( x_1 \)} \\
\text{\( \rightarrow \)} \\
\text{\( x_1 \)}
\end{array}
\]

otherwise,

\[
\begin{array}{c}
\text{\( s \)} \\
\text{\( \rightarrow \)} \\
\text{\( t \)}
\end{array}
\]

\[
\begin{array}{c}
\text{\( x_2 \)} \\
\text{\( \rightarrow \)} \\
\text{\( x_2 \)}
\end{array}
\]
Summary of the Core BiX

\[
X \ ::= \ xid \mid xconst \ S \mid xchild \mid xsetcnt \ X \mid \ldots \\
\mid X_1; X_2 \mid X_1 || X_2 \mid xmap \ X \mid xif \ P \ X_1 \ X_2 \\
\mid xlet \ Var \ X \mid xvar \ Var \\
\mid xfunapp \ fname \ [X_1, \ldots, X_n]
\]

\[
P \ ::= \ xwithtag \ str \mid xistext \mid xiselement \mid X
\]

\[
Def \ ::= \ fun \ fname(Var_1, \ldots, Var_n) = X
\]
A Programming Example: toc

- **Source Data:**

```xml
<book>
  <title>Data on the Web</title>
  <author>Serge</author><author>Peter</author> <author>Dan Suciu</author>
  <section id="intro" difficulty="easy">
    <title>Introduction</title><p>Text ... </p>
    <section>
      <title>Audience</title><p>Text ... </p>
    </section>
  </section>
  <section id="syntaxnew" difficulty="medium">
    <title>A Syntax For Data</title><p>Text ... </p>
  </section>
</book>
```
Target data we want to have is a table-of-contents.

```xml
<toc>
  <section id="intro">
    <title>Introduction</title>
    <section><title>Audience</title></section>
  </section>
  <section id="syntaxnew">
    <title>A Syntax For Data</title>
  </section>
</toc>
```
A Programming Example: toc

The transformation in BiX to produce the table of contents from a book:

```plaintext
fun toc($book-or-section) =
  xvar $book-or-section; xchild;
  xmap (xif (xwithtag 'section') X0 (xconst ()))
where
  X0 = xlet $section
       (xconst <section>[]; xsetcnt (X1||X2))
  X1 = xvar $section; xchild;
       xmap (xif (xwithtag 'title') xid (xconst ()))
  X2 = xfunapp toc [xvar $section]
```
We can obtain BiX from XQuery

- Making the table-of-contents from a book.

```
declare function local:toc($book-or-section)
{
    for $section in $book-or-section/section
    return
        <section>
            {$section/@id, $section/title,
            local:toc($section)}
        </section>
};
<toc>
  { for $s in doc("book.xml")/book
    return local:toc($s)}
</toc>
```
Outline

1. Bidirectional Transformation
2. Bidirectional Transformation in Document Construction
3. BiX: A Bidirectional Transformation Language
4. From Document Engineering to Software Engineering
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!
- Specifically, ABC would be empowered by it!
Software companies ask for it!
Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!

2003.12:
Software companies ask for it!
Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!

2003.12: We thought it at Robust Software Construction Workshop.
Software companies ask for it!
Many researchers in ASE are talking about it!

Specifically, **ABC would be empowered by it!**
- **2003.12:** We thought it at Robust Software Construction Workshop.
- **2006.1:**
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!
- 2003.12: We thought it at Robust Software Construction Workshop.
- 2006.1: We started a joint project (in China) for it.
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!

Specifically, **ABC would be empowered by it!**
  - **2003.12:** We thought it at Robust Software Construction Workshop.
  - **2006.1:** We started a joint project (in China) for it.
  - **2007.4:**
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!

- **2003.12**: We thought it at Robust Software Construction Workshop.
- **2006.1**: We started a joint project (in China) for it.
- **2007.4**: We started a joint project (in Japan) for it.
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!

Specifically, **ABC would be empowered by it!**

- **2003.12:** We thought it at Robust Software Construction Workshop.
- **2006.1:** We started a joint project (in China) for it.
- **2007.4:** We started a joint project (in Japan) for it.
- **2007.11:**
Software Engineering Wants Bidirectional Transformation

- Software companies ask for it!
- Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!

- 2003.12: We thought it at Robust Software Construction Workshop.
- 2006.1: We started a joint project (in China) for it.
- 2007.4: We started a joint project (in Japan) for it.
- 2007.11: We announced our first result in ASE’07.
Software companies ask for it!
Many researchers in ASE are talking about it!

Specifically, ABC would be empowered by it!
- **2003.12**: We thought it at Robust Software Construction Workshop.
- **2006.1**: We started a joint project (in China) for it.
- **2007.4**: We started a joint project (in Japan) for it.
- **2007.11**: We announced our first result in ASE’07.
- **2007.12**:
Software companies ask for it!
Many researchers in ASE are talking about it!

Specifically, **ABC would be empowered by it!**

- **2003.12**: We thought it at Robust Software Construction Workshop.
- **2006.1**: We started a joint project (in China) for it.
- **2007.4**: We started a joint project (in Japan) for it.
- **2007.11**: We announced our first result in ASE’07.
- **2007.12**: We are at our first joint workshop on it.
Bidirectional Software Engineering is Challenging

- Software development requires precise but understandable **semantics** of bidirectional transformations for communication among developers at different stages.
Bidirectional Software Engineering is Challenging

- Software development requires precise but understandable semantics of bidirectional transformations for communication among developers at different stages.
- Transformation in software engineering is mostly applied to graphs instead of trees.
Bidirectional Software Engineering is Challenging

- Software development requires precise but understandable **semantics** of bidirectional transformations for communication among developers at different stages.
- Transformation in software engineering is mostly applied to **graphs** instead of trees.
- Usefulness of bidirectional software engineering cannot be convinced if it is not applied to **large-scale** applications.
Bidirectional Software Engineering is Challenging

- Software development requires precise but understandable semantics of bidirectional transformations for communication among developers at different stages.
- Transformation in software engineering is mostly applied to graphs instead of trees.
- Usefulness of bidirectional software engineering cannot be convinced if it is not applied to large-scale applications.
Two Questions for this Workshop

- What kind of transformation languages are required for describing software development process (from specification to final implementation) in ABC?
Two Questions for this Workshop

What kind of transformation languages are required for describing software development process (from specification to final implementation) in ABC? Is ATL fine, if we do not think about backward transformation?
Two Questions for this Workshop

- What kind of transformation languages are required for describing software development process (from specification to final implementation) in ABC? Is ATL fine, if we do not think about backward transformation?

- Which part of ABC is better to be used as the first experiment of "fusion" with bidirectional transformation?
Two Questions for this Workshop

- What kind of transformation languages are required for describing software development process (from specification to final implementation) in ABC? Is ATL fine, if we do not think about backward transformation?
- Which part of ABC is better to be used as the first experiment of ”fusion” with bidirectional transformation?

software engineering +
programming languages +
transformational programming ⇒ a new paradigm