BiFluX: A Bidirectional Functional Update Language for XML

Hugo Pacheco[1]   Tao Zan[2]   Zhenjiang Hu[2]

[1] Cornell University, USA

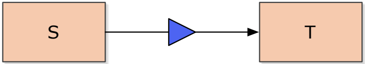
[2] National Institute of Informatics, Tokyo, Japan

PPDP 2014

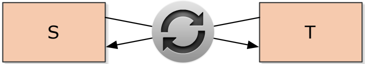
Canterbury, September 10th, 2014

XML Transformation Languages

- XML data formats abound for data exchange and processing
- XML Transformation Languages (XQuery, XSLT, XDuce) ...
- ... are essential to convert data between different formats



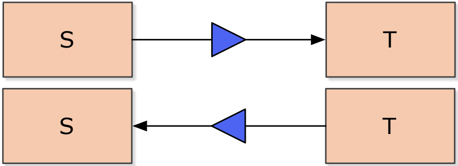
- ... but unsatisfactory to mutually convert between such formats (a maintenance nightmare!)



Bidirectional Transformations (BXs)

"A mechanism for maintaining the consistency of two (or more) related sources of information."

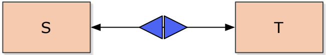
[Czarnecki et al., ICMT 2009]



- many bidirectional transformation approaches support XML formats

BX approaches - Relational

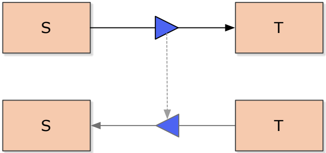
- write a consistency relation between the two schemas in a declarative language
- derive both transformations from the consistency relation



- examples:
  - biXid [Kawanaka & Hosoya, ICFP 2006]
  - XSugar [Brabrand et al., DBPL 2005]
  - QVT [OMG, 2011]

BX approaches - Bidirectionalization

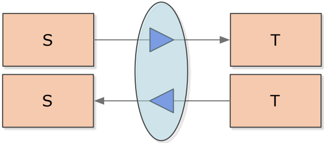
- write a (typically lossy) forward transformation in a common programming language
- derive the backward transformation



- examples:
    - XQuery views [Fegaras, ICDE 2010;Liu et al., PEPM 2007]
    - polymorphic Haskell functions [Matsuda & Wang, PPDP 2013]

Bx approaches - Combinatorial

- write a program in a domain-specific bidirectional language
- each program denotes both transformations
- composition; correct-by-construction



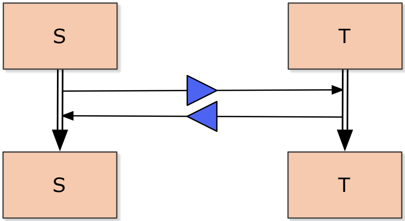
- examples:
  - Focal [Foster et al., TOPLAS 2007]
  - X [Hu et al., PEPM 2004]
  - Multifocal [Pacheco & Cunha, ICMT 2012]
  - etc

BX approaches - Current Picture

- due to the latent ambiguity of BXs
- existing approaches focus mainly on enforcing consistency
- from the programmer's perspective, they suffer either from:
  - supporting only "trivial" BXs
  - being unpredictable, by making arbitrary choices and giving little control over what the BX does
  - being impractical to specify complex BXs

BXs = Updates

> *"Intuitively, a BX translates updates on a source model into updates on a target model, and vice-versa, so that the updated models are kept consistent."*



XML Update Languages

- XML transformation languages (XQuery, XSLT, XDuce) are bad for specifying small updates
- a few dedicated languages for in-place XML updates:
  - XQuery Update Facility [W3C, 2011]:
    - imperative language
    - ill-understood semantics (aliasing, side-effects, depends on traversal order)
  - Flux (Functional Lightweight Updates for XML) [Cheney, ICFP 2008]:
    - functional language
    - clear semantics
    - static typing
    - straightforward type-checking
  - XUpdate, XQuery!, and many others...

A Flux Example

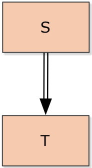
```
UPDATE books/book BY
        INSERT AS LAST INTO author
        VALUE 'Stephen Buxton'
WHERE title = 'Querying XML'
```

$$\begin{aligned} & \mathit{books}\,[\mathit{book}\,[\mathit{author}\,[\mathtt{string}], \mathit{title}\,[\mathtt{string}]]*] \\ \rightarrow\ & \mathit{books}\,[\mathit{book}\,[\mathit{author}\,[\mathtt{string}]+, \mathit{title}\,[\mathtt{string}]]*] \end{aligned}$$

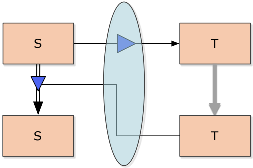
Our proposal: BiFluX

- We propose BiFluX, a bidirectional variant of Flux
- particular class of BXs: lenses, view updating
- modest syntactic extension
  - notion of view (feat. pattern matching, non-in-place updates)
  - static restrictions to ensure well-behavedness

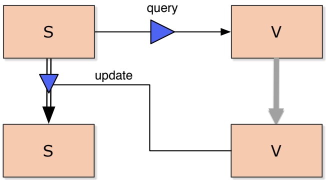
- Flux: unidirectional *in-place* semantics



- BiFluX: bidirectional *view-update* semantics



BiFluX - A Bidirectional Update Language



- a bidirectional update says:
  - which parts of the source are to be updated
  - how view modifications are reflected to the source
- there is a unique query function for each BiFluX program
- consistency properties of lenses [Foster et al., TOPLAS 2007]:

$$
\begin{array}{rcll}
\mathit{Update}(s, v') = s' & \Rightarrow & \mathit{Query}(s') = v' & \textsc{UpdateQuery} \\
\mathit{Query}(s) = v & \Rightarrow & \mathit{Update}(s, v) = s & \textsc{QueryUpdate}
\end{array}
$$

A BiFluX example (1)

Is this a bidirectional *update*?

```
UPDATE $source/books/book BY
        INSERT AS LAST INTO author
        VALUE $view
WHERE SOURCE title = 'Querying XML'
```

$$S = \mathit{books}\,[\mathit{book}\,[\mathit{author}\,[\texttt{string}]+, \mathit{title}\,[\texttt{string}]]*]$$
$$V = \texttt{string}$$

A BiFluX example (1)

Is this a bidirectional *update*?

```
UPDATE $source/books/book BY
        INSERT AS LAST INTO author
        VALUE $view
WHERE SOURCE title = 'Querying XML'
```

$$S = \mathit{books}\,[\mathit{book}\,[\mathit{author}\,[\texttt{string}]+, \mathit{title}\,[\texttt{string}]]*]$$
$$V = \texttt{string}$$

- adds the view as the last author to the source authors
- violates the QueryUpdate consistency law!

A BiFluX example (2)

Is this a bidirectional *update*?

```
UPDATE $source/books/book BY
        REPLACE IN author[last()]
        WITH $view
WHERE SOURCE title = 'Querying XML'
```

$$\begin{aligned} S &= \mathit{books}\;[\mathit{book}\;[\mathit{author}\;[\texttt{string}]+, \mathit{title}\;[\texttt{string}]]*] \\ V &= \texttt{string} \end{aligned}$$

A BiFluX example (2)

Is this a bidirectional *update*?

```
UPDATE $source/books/book BY
        REPLACE IN author[last()]
        WITH $view
WHERE SOURCE title = 'Querying XML'
```

$$S = \mathit{books}\ [\mathit{book}\ [\mathit{author}\ [\texttt{string}]+, \mathit{title}\ [\texttt{string}]]*]$$
$$V = \texttt{string}$$

- replaces the last author in the source with the view author
- well-behaved bidirectional update!

XML types and values

- XDuce-style regular expression types [Hosoya et al., TOPLAS 2005] (with *n*-guarded recursion)

$$\begin{aligned}\alpha &::= \texttt{bool} \parallel \texttt{string} \parallel n[\tau] \\ \tau &::= \alpha \parallel () \parallel \tau \mid \tau' \parallel \tau, \tau' \parallel \tau^* \parallel X\end{aligned}$$

- Flux: "flat" representation of values as trees/forests
  - economical, hard to embed into functional languages w/o structural type equivalence

$$\begin{aligned}ft &::= \texttt{true} \mid \texttt{false} \mid w \mid n[fv] \\ fv &::= () \mid ft, fv\end{aligned}$$

- BiFluX: structured representation of values as ADTs
  - "witness how to parse a flat value as an instance of a type"

$$\begin{aligned}t &::= \texttt{true} \mid \texttt{false} \mid w \mid n[v] \\ v &::= t \mid () \mid L\ v \mid R\ v \mid (\ v, v\ ) \mid [\ v_0, \ldots, v_n\ ]\end{aligned}$$

XML Subtyping

- Flux: type-checking with inclusion-based subtyping

$$\tau <: \tau' \text{ \emph{iff} } [\![\tau]\!]_{\mathit{flat}} \subseteq [\![\tau']\!]_{\mathit{flat}}$$

- equivalence relation that ignores structure

$$v \sim v' \triangleq \mathit{flat}(v) = \mathit{flat}(v')$$

- BiFluX: we need more than subtyping
- we reuse an algorithm with additional witness functions between underlying structured values [Lu and Sulzmann, APLAS 2004]

$$\begin{tikzcd}
\tau \arrow[r, bend left, "ucast"] \arrow[r, phantom, "<:"] & \tau' \arrow[l, bend left, "dcast"]
\end{tikzcd}$$

$$\mathit{ucast}\ v \sim v \qquad \textsc{Up}_{\sim}$$

$$\mathit{dcast}\ v' = v \Rightarrow v \sim v' \qquad \textsc{Down}_{\sim}$$

Core Language

- BiFluX → core language
- we consider two kinds of core updates and semantics
  - bidirectional semantics as *lenses*

    Hugo Pacheco and Zhenjiang Hu and Sebastian Fischer
    Monadic Combinators for "Putback" Style Bidirectional Programming
    PEPM 2014.

  - unidirectional semantics as *arrows*

    James Cheney
    Flux: FunctionaL Updates for XML
    ICFP 2008.

- core BiFluX language (novelties in green):

e ::= "core XQuery expressions"

p ::= "simple XPath expressions"

pat ::= "linear, sequence-based XDuce patterns"

u ::= "Flux unidirectional updates"

b ::= "BiFluX bidirectional updates"

Core language: Unidirectional updates

- Flux in-place updates u modify specific parts of the source and leave the remaining data unchanged
- purely value-based semantics

$$\gamma; v \vdash u \Rightarrow v'$$

"in environment γ and focus v, the unidirectional update u updates v to value v'"

- independent typing

$$\Gamma \vdash \{\tau\}\, u\, \{\tau'\}$$

"in type environment Γ, the unidirectional update u maps values of type τ to values of type τ'"

Core language: Bidirectional updates

- BiFluX bidirectional updates b are interpreted as:
  - an *update* function that modifies specific parts of the source to embed all view information
  - a *query* function that computes a view of a given source
- semantics is given to type derivations

$$\Gamma \vdash \{\tau_S\}\, b\, \{\tau_V\} \Rightarrow (query, udpate)$$

"in type environment Γ, the bidirectional update b defines a BX $(query, update)$ between source type τ_S and view type τ_V, with $query : \tau_S \rightarrow \tau_V$ and $update : \Gamma \rightarrow \tau_S \rightarrow \tau_V \rightarrow \tau_S$

BiFluX Syntax

- BiFluX high-level language (changes to Flux in green):

```
Stmt     ::=  Upd [WHERE Conds] | Stmt ; Stmt | { Stmt } | { }
           |  IF Tag Expr THEN Stmt ELSE Stmt
           |  LET Tag Pat = Expr IN Stmt
           |  CASE Tag Expr OF { Cases }
Upd      ::=  INSERT (BEFORE | AFTER) PatPath VALUE Expr
           |  INSERT AS (FIRST | LAST) INTO PatPath VALUE Expr
           |  DELETE [FROM] PatPath | REPLACE [IN] PatPath WITH Expr
           |  UPDATE PatPath BY Stmt
           |  UPDATE PatPath BY VStmt FOR VIEW PatPath [Match]
           |  KEEP PatPath | CREATE VALUE Expr
Conds    ::=  Tag Expr [; Conds] | Tag Var := Expr [; Conds]
Cases    ::=  Pat → Stmt | Cases '|' Cases
VStmt    ::=  { VStmt } | VUpd
           |  VUpd '|' VUpd
VUpd     ::=  MATCH → Stmt
           |  UNMATCHS → Stmt
           |  UNMATCHV → Stmt
Match    ::=  MATCHING BY Path
           |  MATCHING SOURCE BY Path
                      VIEW BY Path
PatPath  ::=  [Pat IN] Path
Tag      ::=  [SOURCE | VIEW]
```

A *bookstore* BiFluX Example

```
UPDATE $book IN $source/bookstore/book BY
{
  MATCH -> REPLACE price WITH $price
| UNMATCHV -> CREATE VALUE <book category=’undefined’>
                             <title/>
                             <author>??</author>
                             <year>??</year>
                             <price/>
                          </book>
}
FOR VIEW book[$title AS v:title, $price AS v:price] IN $view/books/*
MATCHING SOURCE BY $book/title VIEW BY $title
```

A *bookstore* BiFluX Example: Forward

- Source:

```
<bookstore>
 <book>
   <title >Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
 </book>
 <book category='Programming'>
   <title >Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
 </book>
</bookstore>
```

- View:

```
<books>
 <book>
   <title>Harry Potter</title>
   <price>29.99</price>
 </book>
 <book>
   <title>Learning XML</title>
   <price>39.95</price>
 </book>
</books>
```

A *bookstore* BiFluX Example: View update

- Source:

```
<bookstore>
 <book>
   <title >Harry Potter</title>
   <author>J K. Rowling</author>
   <year>2005</year>
   <price>29.99</price>
 </book>
 <book category=’Programming’>
   <title >Learning XML</title>
   <author>Erik T. Ray</author>
   <year>2003</year>
   <price>39.95</price>
 </book>
</bookstore>
```

- Updated View:

```
<books>
 <book>
  <title>XPath for Dummies</title>
  <price>19.99</price>
 </book>
 <book>
   <title>Harry Potter</title>
   <price>19.99</price>
 </book>
 <book>
   <title>Learning XML</title>
   <price>19.99</price>
 </book>
</books>
```

A *bookstore* BiFluX Example: Backward

- Updated Source:

```
<bookstore>
 <book category=’undefined’>
   <title>XPath for Dummies</title>
   <author>??</author> <year>??</year>
   <price>19.99</price>
 </book>
 <book>
   <title>Harry Potter</title>
   <author>J K. Rowling</author> <year>2005</year>
   <price>19.99</price>
 </book>
 <book category=’Programming’>
   <title>Learning XML</title>
   <author>Erik T. Ray</author> <year>2003</year>
   <price>19.99</price>
 </book>
</bookstore>
```

Conclusions

- proposed a novel bidirectional programming by update approach
  - declarative style (write an update)
  - good configurability (direct control over the update strategy)
- presented BiFluX, a bidirectional XML update language
- I hope to have convinced you that BiFluX allows users to write BXs in a friendly notation and at a nice level of abstraction
- type-safe, strongly-typed implementation in Haskell
- for demos, our tool and more BiFluX examples see...

`http://www.prg.nii.ac.jp/projects/BiFluX`