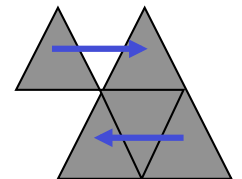
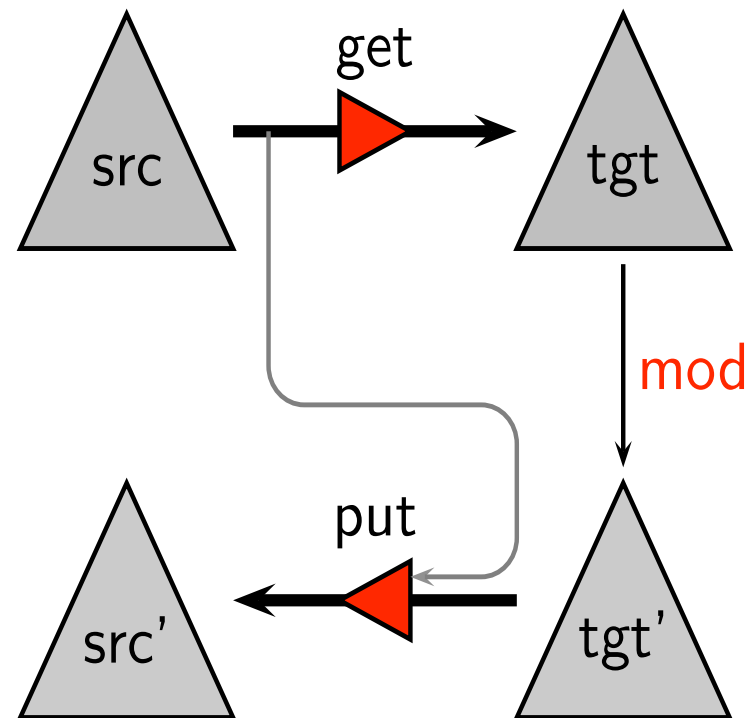
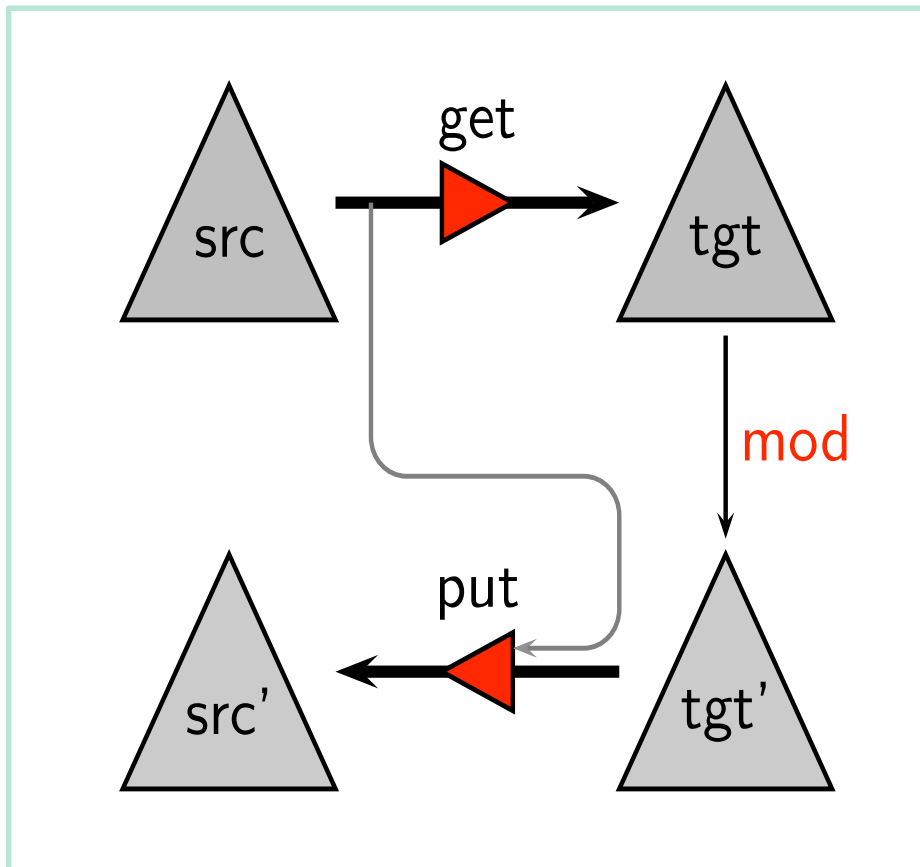


# Bidirectional Transformation (BX)

[Nate Foster, et al: POPL 2005]



# Roundtrip Properties

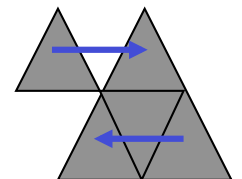


Get-Put:

$$\text{put } s (\text{get } s) = s$$

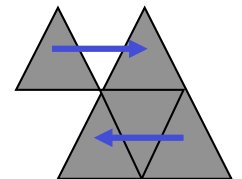
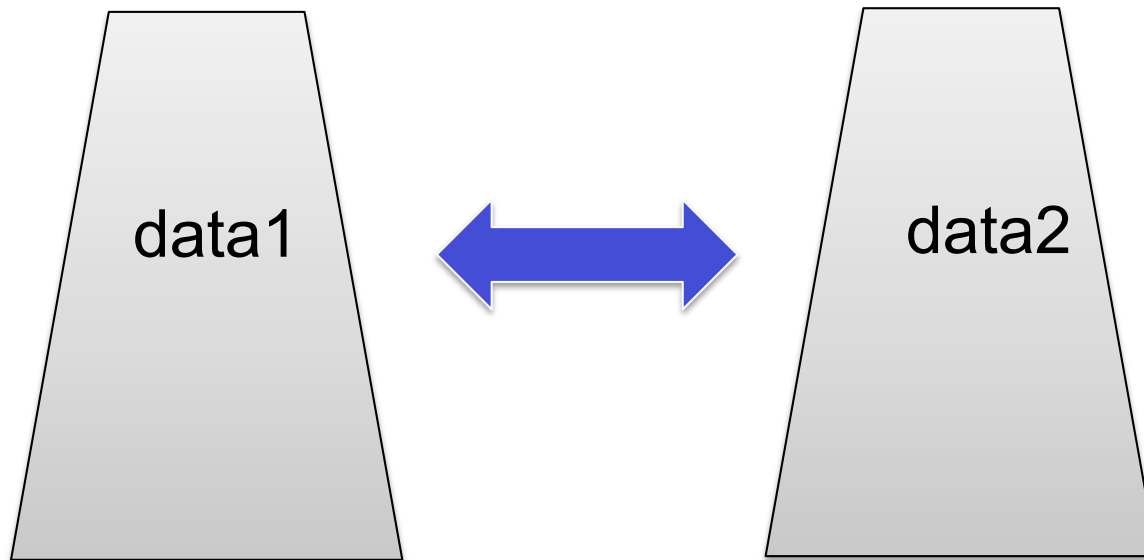
Put-Get:

$$\text{get } (\text{put } s \ t) = t$$



# What is BX Programming?

Define **a pair of functions get/put** such that the two kinds of data can be synchronized.



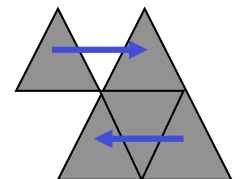
# Bidirectional Transformation Languages



- 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/UnQL+** Univ. of Tokyo / NII  
[PEPM'04, ICFP'07, ASE'07, FSE'09, ESOP'10, ICFP'10, MODELS'10, PPDP'11, ICSE'12, PPDP'13, ICFP'13]



# Existing Approaches to Bidirectional Programming

“get” (forward transformation)

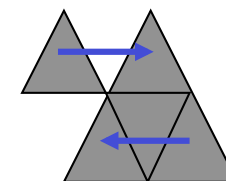


“put” (backward transformation)

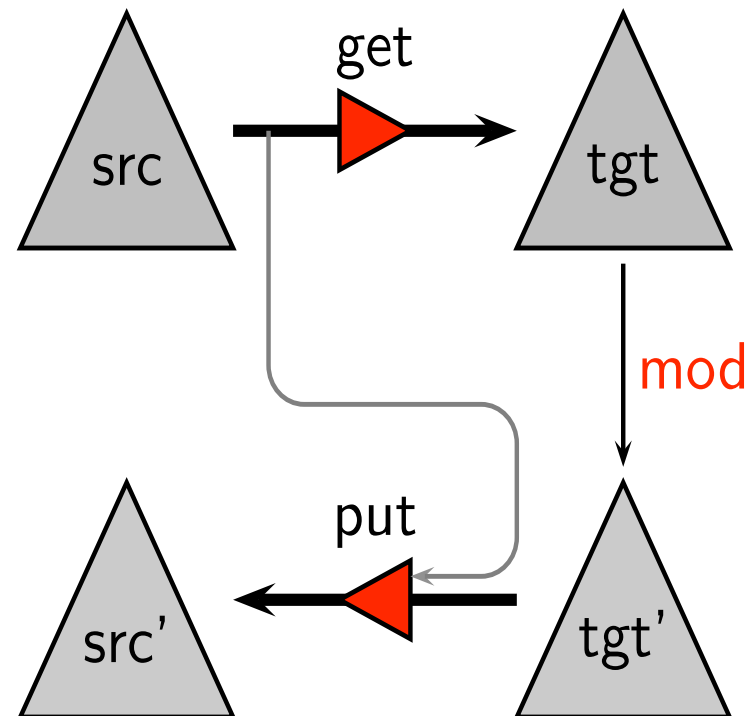
- Domain Specific Bidirectional Languages
- Automatic Bidirectionization of ATL, XQuery, UnQL

**Assumption:**

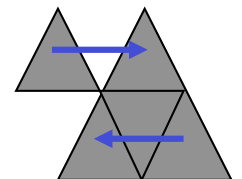
Each get has one corresponding put.



# Not Practical!

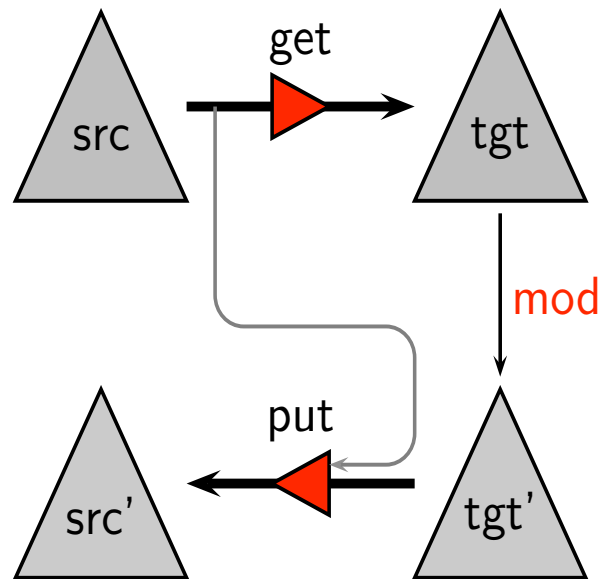


Since get is generally **non-injective**, many suitable puts correspond to one get, each being useful in different context.



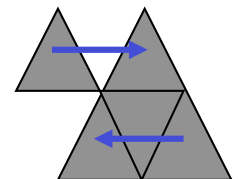
# Put is the essence of BX!

- An important but little-known fact:



**put** uniquely determines **get**.

Sebastian Fischer, Zhenjiang Hu, Hugo Pacheco, A Clear Picture of Lenses, submitted to JFP, 2013 (available upon request)

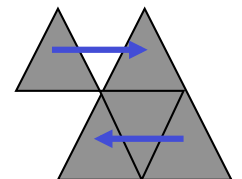


# Put-based BX Programming

Define **a pair of functions get/put** such that the two kinds of data can be synchronized.



Define **a well-behaved put** such that the two kinds of data can be synchronized.

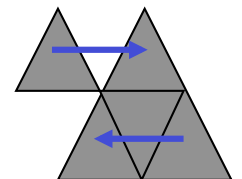




# This talk

Can we provide a language for programming  
“put”?

- Easy to use
- Powerful to specify various “put”
- Static check of well-behavedness of “put”
- Automatic derivation of “get”



# What is the essence of “put”?

$\text{put} : \text{Old Source} \rightarrow \text{View} \rightarrow \text{New Source}$

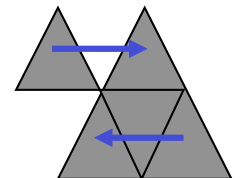


$\text{put}' : \text{View} \rightarrow \text{Old Source} \rightarrow \text{New Source}$



$\text{put}' : \text{View} \rightarrow (\text{Old Source} \rightarrow \text{New Source})$

“put” uses a view to update the source.

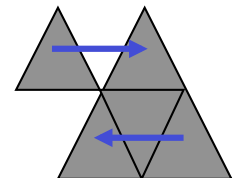


“put” is nothing but an update language!

- Languages for updating XML trees
  - XQuery!
  - Flux [ICFP 2008]: a functional update language
- There are huge number of update languages, general and specific

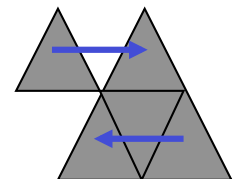


BiFlux: A View-Embedding Update Language



- Flux (Functional Lightweight Updates for XML)  
[Cheney, ICFP 2008]:
  - functional language
  - clear semantics
  - straightforward type-checking

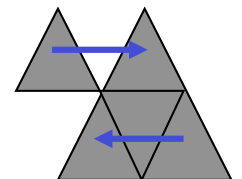
```
UPDATE $x AS books/book BY  
  REPLACE IN year WITH "1859"  
  INSERT ...  
  DELETE ...  
  UPDATE .... BY ...  
  ...  
WHERE $x/title/text() = "A Tale of Two Cities"
```



- Use an XML view to update an XML source
  - All the view must be embedded in the updated source

UPDATE source BY  
Update Stmt1  
...  
FOR VIEW view  
matching-clause

MATCH → Stmt  
UNMATCHS → Stmt  
UNMATCHV → Stmt



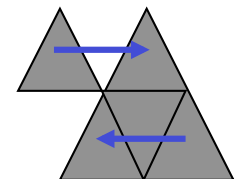
# A Bookstore Example

## Source:

```
<bookstore>
  <book category='Food'>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

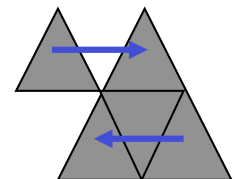
## View:

```
<books>
  <book>
    <title>Everyday Italian</title>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <price>29.99</price>
  </book>
</books>
```



# Update Bookmark with View

```
PROCEDURE updateBookStore(source $source AS s:bookstore, view $view AS v:books) =  
UPDATE $book IN $source/bookstore/book BY  
{  
    MATCH    -> REPLACE price WITH $price  
    UNMATCHV -> CREATE VALUE <book category='undefined'  
                        <title/>  
                        <author>??</author>  
                        <year>??</year>  
                        <price/>  
                        </book>  
    UNMATCHS -> DELETE .  
}  
FOR VIEW book[$title AS v:title, $price AS v:price] IN $view/books/*  
MATCHING SOURCE BY $book/title VIEW BY $title
```



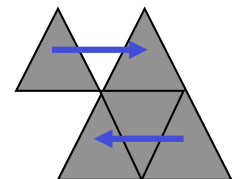
# Update Books when Titles Match

## Source:

```
<bookstore>
  <book category='Food'>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```





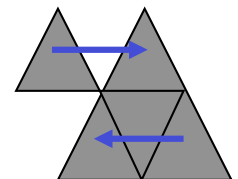
# Update Books when Titles Match

## Source:

```
<bookstore>
  <book category='Food'>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>19.99</price>
  </book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



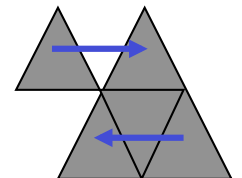
# Update Books with no Corresponding Books in the View

## Source:

```
<bookstore>
  <book category='Food'>
    <title>Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book>
    <title>Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>19.99</price>
  </book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



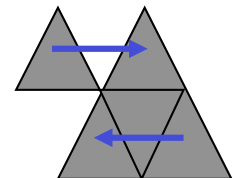
# Update Books with no Corresponding Books in the View

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>19.99</price>
</book>
</bookstore>
```

## View':

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



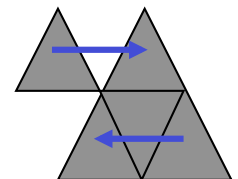
# Update Books with Books only in the View

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>19.99</price>
</book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



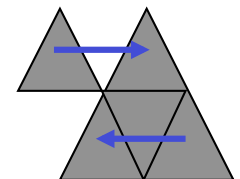
# Update Books with Books only in the View

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>19.99</price>
</book>
<book category='undefined'>
  <title />
  <author>??</author>
  <year>??</year>
  <price/>
</book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



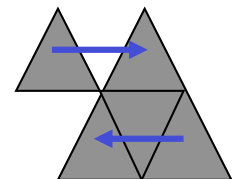
# Update Books with Books only in the View

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>19.99</price>
</book>
<book category='undefined'>
  <title>XQuery Kick Start</title>
  <author>??</author>
  <year>??</year>
  <price>29.99</price>
</book>
</bookstore>
```

## View:

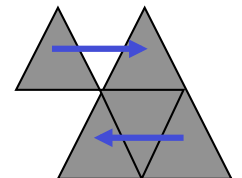
```
<books>
  <book>
    <title>Harry Potter</title>
    <price>19.99</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



# Forward Update Propagation for Free

Source:

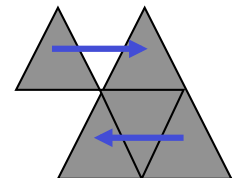
```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>19.99</price>
</book>
<book category='undefined'>
  <title>XQuery Kick Start</title>
  <author>??</author>
  <year>??</year>
  <price>29.99</price>
</book>
</bookstore>
```



# Forward Update Propagation for Free

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>100.00</price>
</book>
<book category='Computer Science'>
  <title>XQuery Kick Start</title>
  <author>Taro Tanaka</author>
  <year>2010</year>
  <price>29.99</price>
</book>
</bookstore>
```





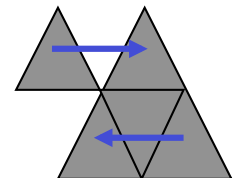
# Forward Update Propagation for Free

## Source:

```
<bookstore>
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>100.00</price>
</book>
<book category='Computer Science'>
  <title>XQuery Kick Start</title>
  <author>Taro Tanaka</author>
  <year>2010</year>
  <price>29.99</price>
</book>
</bookstore>
```

## View:

```
<books>
  <book>
    <title>Harry Potter</title>
    <price>100.00</price>
  </book>
  <book>
    <title>XQuery Kick Start</title>
    <price>29.99</price>
  </book>
</books>
```



# BiFlux Demo (by Tao Zan)

An update-based approach for bidirectional model transformation - Mozilla Firefox

localhost:3000

## BiFluX

ABSTRACT DEMONSTRATION SOURCES

### An update-based bidirectional model transformation framework

Abstract Framework Syntax Test Cases Source

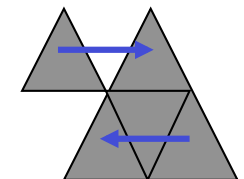
#### Abstract

Bidirectional model transformations play an important role in model-driven software development for maintaining the consistency between two models. All existing approaches are defining either relations or mapping functions between source and target models. As forward transformation is not bijective, backward transformation is not unique. Currently the ambiguous of backward transformation is solved in all the existing systems by deriving a default one from forward transformation. While programmers have no control over backward transformation. Even though the consistency between two models is kept, in fact they are not truly correct. Our solution follows a pragmatically different approach from existing bidirectional model transformation approaches: instead of writing forward transformations, programmers write backward transformations by specifying how a target model can be used to update a source model. The novel approach allows programmers to transparently control how updates can be reflected back to a source model. While under certain conditions, we guarantee that there exists only one forward transformation combining with the backward transformation satisfy the correctness which can be derived from our system.

#### Framework Overview

The figure on the right side depicts the architecture of our update-based bidirectional model transformation framework. A BiFluX update program is evaluated in two stages. On a first stage, it is statically compiled against a source and a view meta-model (represented as DTDs), to produce a bidirectional executable. The generated executable can then be

```
graph TD; Source[Source DTD] --> Compiler[Bidirectional Compiler]; Target[Target DTD] --> Compiler; Program[BiFluX Program] --> Compiler;
```

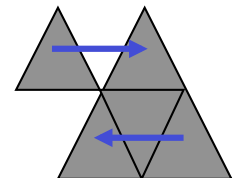


## Conclusion

We design and implement BiFlux, an update language for programming “put”:

- Easy to use
- Powerful to specify various “put”
- Static check of well-behavedness of “put”
- Automatic derivation of “get”

A post-doc position is open.



# BiG Project Web Site



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

