Bidirectional Data Transformation by Calculation

Hugo Pacheco

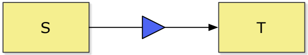
HASLab
INESC TEC & Universidade do Minho, Braga, Portugal

September 17th 2012
Braga

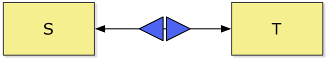
Outline

Data Transformations

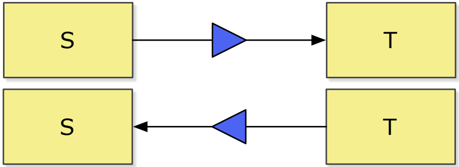
- data transformations abound in software engineering
- essential to convert data between different formats



- in real model-driven software engineering scenarios, we often need to run a transformation in both directions



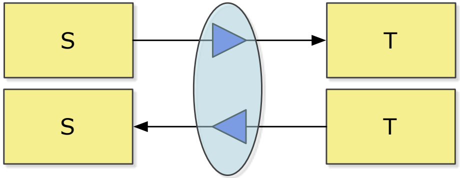
(*Ad hoc*) Bidirectional Transformations



Manual design: two separate transformations

- expensive
- error-prone
- a maintenance problem

Bidirectional Languages

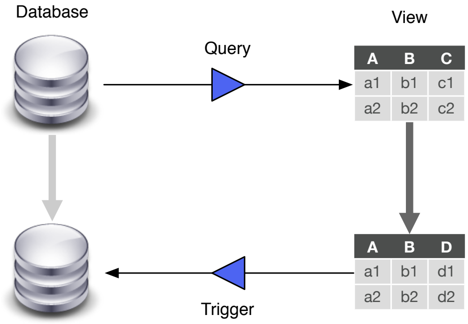


Combinatorial design: the same specification denotes both

- nice syntax
- clean semantics
- compositional

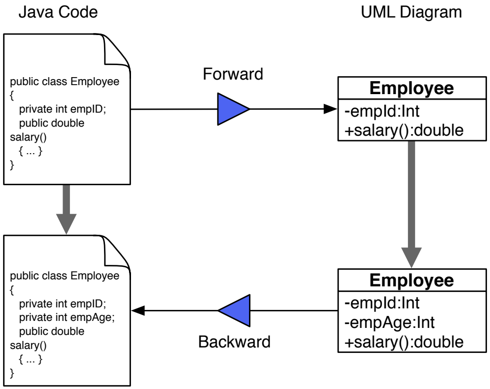
Bidirectional Languages exist for ...

...databases...



Bidirectional Languages exist for ...

...model-driven software engineering...

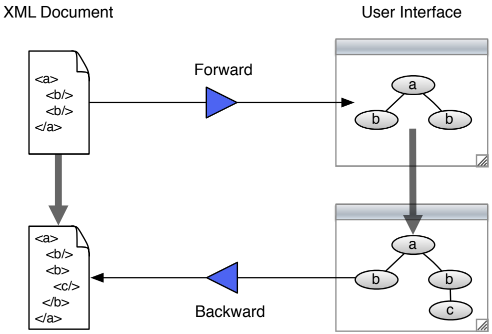


```
public class Employee
{
   private int empID;
   public double
salary()
   { ... }
}
```

```
public class Employee
{
   private int empID;
   private int empAge;
   public double
salary()
   { ... }
}
```

Bidirectional Languages exist for ...

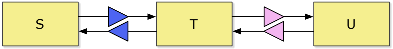
...user interfaces...



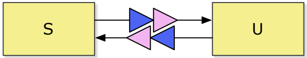
...etc

Motivation - Efficiency

- combinatorial approaches build complex transformations by composition

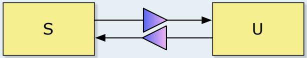


- composition ⇒ cluttering ⇒ inefficiency



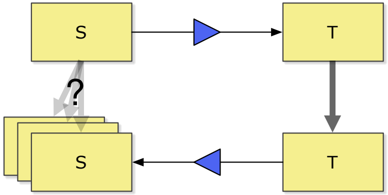
Question

- how to optimize bidirectional transformations?



Motivation - Configurability

- for non-bijective transformations, an update may have many corresponding updates

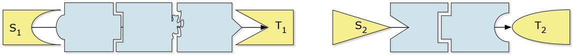


Question

- how to allow users to choose a suitable update?

Motivation - Genericity

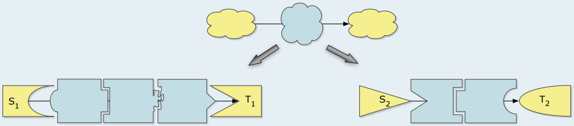
- bidirectional transformations are typically built to match a specific structure, via multiple steps
- for the same high-level transformation, different low-level transformations must be built to handle different structures



- impractical to write complex transformations
- does not support evolution

Question

- how to define a transformation in a generic and concise way?

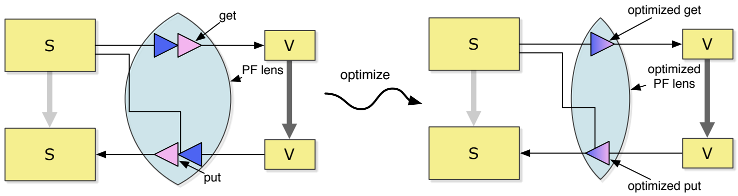


State of the Art

- vast number of approaches for various purposes
- hard to classify and compare different approaches:
  - understand their advantages and limitations
  - assess progress on the field
- we propose a taxonomy for the classification of bidirectional approaches
  - Scheme: Framework, Update representation
  - Properties: Round-tripping, Consistency, Totality
  - Deployment: Data domain, Typing, Specification, Language, Bidirectionalization approach
- we survey up to 40 existing approaches

Efficiency ⇒ Point-free Lenses

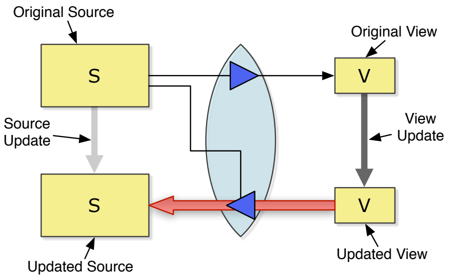
- Foster et al. proposed the bidirectional framework of lenses
- we define a language of point-free bidirectional lenses



- Data domain: algebraic data types (e.g. lists, trees)
- Language: categorical functional programming combinators with no variables
- Algebraic laws: allow to prove properties & to optimize lenses

Configurability ⇒ Point-free Delta Lenses

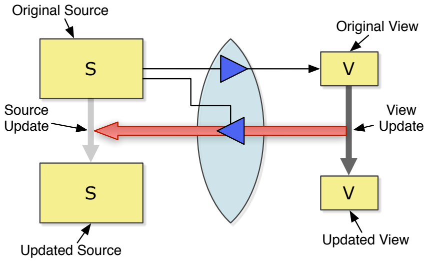
State-based framework: *put* takes the modified view state



- no information about the actual update
- *put* has to “guess” the intended change of the update

Configurability ⇒ Point-free Delta Lenses

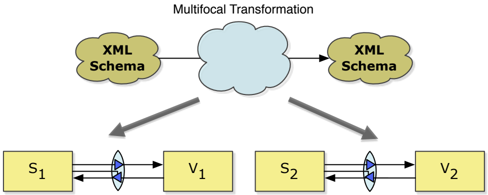
Operation-based framework: *put* takes some knowledge about the view update



- Diskin et al. proposed a conceptual framework of delta lenses
- we define a language of point-free delta lenses
- users can control the choice of a translated source update by giving a precise description of the view update

Genericity ⇒ The *Multifocal* Framework

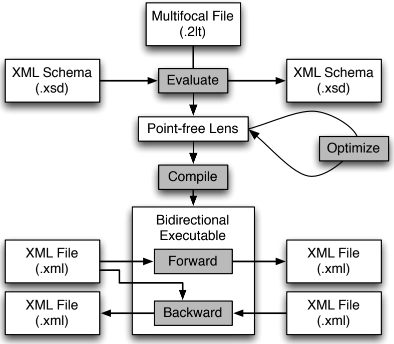
- Foster et al. developed the *Focal* tree transformation language
- we propose the *Multifocal* XML transformation language



- Two-level: from a generic schema-level transformation we get instance-level XML document transformations
- Strategic: concise specification style (e.g. traversals)
- Bidirectional: underlying instance-level transformations as lenses

Genericity $\Rightarrow$ The *Multifocal* Framework

- we implement the *Multifocal* framework



- three stages:
  1. evaluate: XML Schema $\Rightarrow$ XML Schema + lens
  2. optimize (optional): lens $\Rightarrow$ optimized lens
  3. compile: (optimized) lens $\Rightarrow$ executable

Summary

- A detailed picture of the state of the art: taxonomy + survey
- A point-free lens language

  Hugo Pacheco and Alcino Cunha
  Generic Point-free Lenses
  MPC 2010.

Point-free lens library

`http://hackage.haskell.org/package/pointless-lenses`

- An algebraic theory of point-free lenses

  Hugo Pacheco and Alcino Cunha
  Calculating with lenses: optimising bidirectional transformations
  PEPM 2011.

Point-free rewriting library

`http://hackage.haskell.org/package/pointless-rewrite`

Summary

- A point-free delta lens language

  Hugo Pacheco, Alcino Cunha and Zhenjiang Hu
  Delta Lenses over Inductive Types
  BX 2012.

- The *Multifocal* language and framework

  Hugo Pacheco and Alcino Cunha
  Multifocal: A Strategic Bidirectional Transformation Language for XML Schemas
  ICMT 2012.

  Alcino Cunha and Hugo Pacheco
  Algebraic Specialization of Generic Functions for Recursive Types
  Electronic Notes in Theoretical Computer Science, 2011.

Multifocal system

`http://hackage.haskell.org/package/multifocal`

Strategic two-level lens library

`http://hackage.haskell.org/package/pointless-2lt`

Future Work

- Multifocal Framework
  - more expressiveness (language features, bidirectional schemes)
  - more usability (XML integration, empirical study)
- Open BX Challenges
  - unpredictability $\Rightarrow$ new bidirectional properties
    - minimal update translation properties $\Rightarrow$ predictability
  - new operation-based approaches
    - generation of minimal updates
    - deterministic bidirectional programming
  - explore the design space
    - novel frameworks
    - more classification features
    - more complete, precise and self-contained survey