Multifocal: A Strategic Bidirectional Transformation Language for XML Schemas

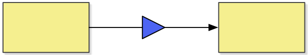
⌜Hugo Pacheco⌟ Alcino Cunha

HASLab / INESC TEC & Universidade do Minho, Braga, Portugal

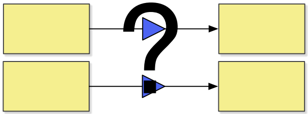
ICMT 2012
Prague - May 28th 2012

Two-level Transformations

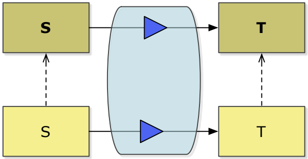
- model transformations are frequent in software engineering



- coupled transformations

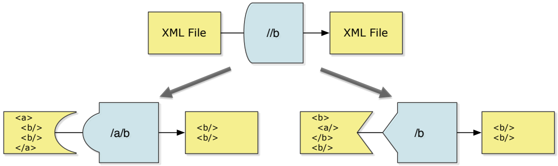


- two-level transformations

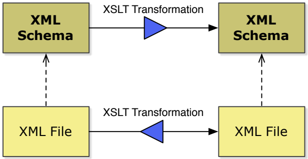


XML Transformation Languages

- XML transformation languages (XSLT, XQuery, XPath)
- generic, structure-shy programs

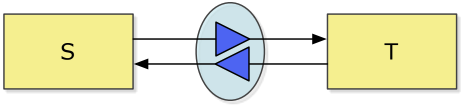


- easier to write, multiple inputs
- not two-level



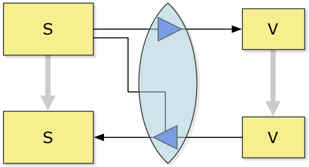
Bidirectional Transformations

- bidirectional transformations

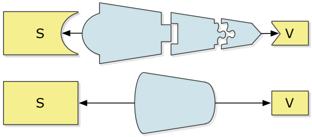


Bidirectional XML Transformation Languages

- many bidirectional languages
- tree-structured data (XML)
- lenses (view-update)

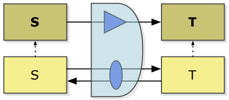


- not two-level, not generic

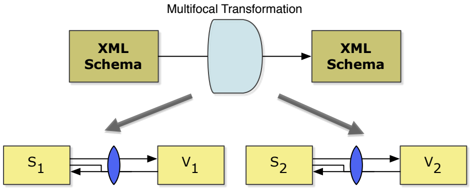


Motivation: Multifocal Language

- two-level bidirectional transformations



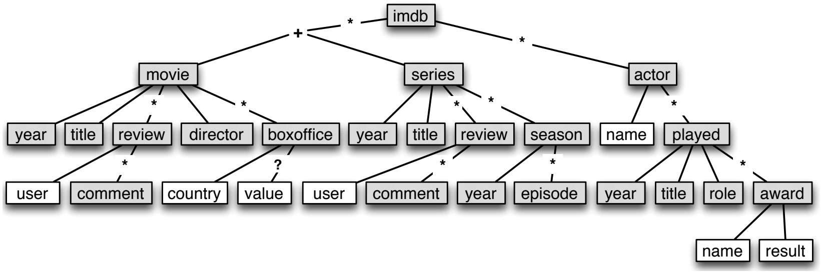
- Multifocal XML transformation language



- schema-level transformations as views between XML Schemas
- model-level transformations as lenses between XML documents
- multiple focus points

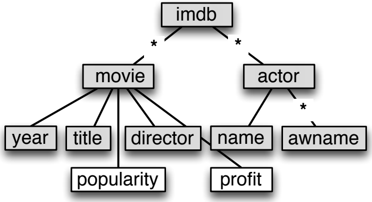
Application Example: XML Views

- source XML Schema modeling a movie database



Application Example: XML Views

- informal XML Schema transformation
  1. delete series
  2. for each movie:
     - count its popularity (total number of review comments)
     - estimate its profit (sum of the boxoffice values)
  3. for each actor, select its name and a list of award names
- view XML Schema



Application Example: XML Views

```
<imdb>
 <movie>
  <year>2003</year>
  <title>Kill Bill: Vol. 1</title>
  <review user="emma">
   <comment>Gorgeous!</comment></review>
  <director>Quentin Tarantino</director>
  <boxoffice country="USA" value="22089322"/>
  <boxoffice country="Japan" value="3521628"/>
 </movie>
 <series><year>2011</year>
  <title>Game of Thrones</title>
  <season><year>2011</year>
   <episode>Winter is Coming</episode>
 </season></series>
 <actor name="Umma Thurman">
  <played><year>2003</year>
   <title>Kill Bill: Vol. 1</title>
   <role>The Bride</role>
   <award name="Saturn" result="Won"/>
 </played></actor>
</imdb>
```

Application Example: XML Views

```
<imdb>
 <movie popularity="1" profit="25610950">
  <year>2003</year>
  <title>Kill Bill: Vol. 1</title>
  <director>Quentin Tarantino</director>
 </movie>
 <actor name="Umma Thurman">
  <awname>Saturn</awname>
 </actor>
</imdb>
```

Application Example: XML Views

```
<imdb>
 <movie> ... </movie>
 <movie popularity="2" profit="15">
  <year>2012</year>
  <title>Sherlock Holmes: Game of Shadows</title>
  <director>Guy Ritchie</director>
 </movie>
 <actor name="Uma Thurman">
  <awname>Saturn Best Actress</awname>
 </actor>
</imdb>
```

Application Example: XML Views

```
<imdb>
 <movie> ... </movie>
 <series> ... </series>
 <movie><year>2012</year>
  <title>Sherlock Holmes: Game of Shadows</title>
  <review user="" comment=""/>
  <review user="" comment=""/>
  <director>Guy Ritchie</director>
  <boxoffice country="" value="15"/>
 </movie>
 <actor name="Uma Thurman">
  <played><year>2003</year>
   <title>Kill Bill: Vol. 1</title>
   <role>The Bride</role>
   <award name="Saturn Best Actress" result="Won"/>
 </played></actor>
</imdb>
```

Multifocal Language: Basic Combinators

- generic style = concise specification
- strategic rewrite system

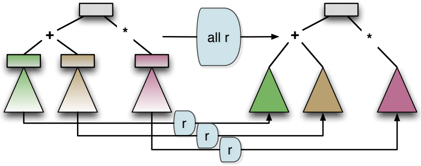
$$Rule = Schema \to Maybe\ (Schema, Lens)$$

- construct flexible strategies in a compositional way
- basic combinators (in what order? how often?)

identity	`nop` : $Rule \to Rule$
sequentially	`(>>)` : $Rule \to Rule \to Rule$
alternatively	`(||)` : $Rule \to Rule \to Rule$
repetitively	`many` : $Rule \to Rule \to Rule$
optionally	`try` : $Rule \to Rule$

Multifocal Language: Traversal Combinators

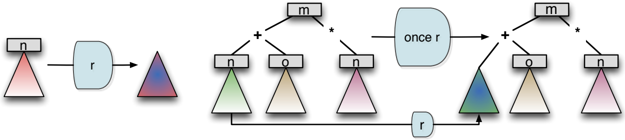
- traversal combinators (at what depth?)
  - apply a rule to all children



  - apply a rule to all descendants

    `everywhere` : *Rule* → *Rule*

  - apply a rule once at an arbitrary depth

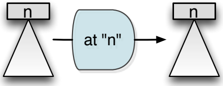


  - apply a rule many times at an arbitrary depth

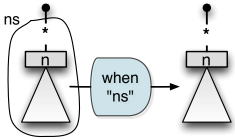
    `outermost` : *Rule* → *Rule*

Multifocal Language: Local Combinators

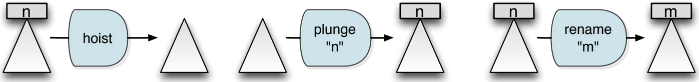
- control the application of certain rules
- local combinators (under which conditions?)
  - at a particular element



  - at a particular location



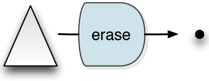
- XML name-based combinators



Multifocal Language: Abstraction Combinators

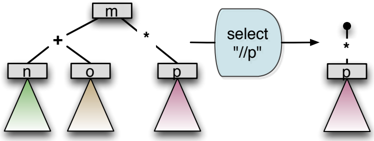
- language for defining XML views
- abstraction combinators

- erase the current tree (explicit)



- empty tree

- apply an XPath query (implicit)

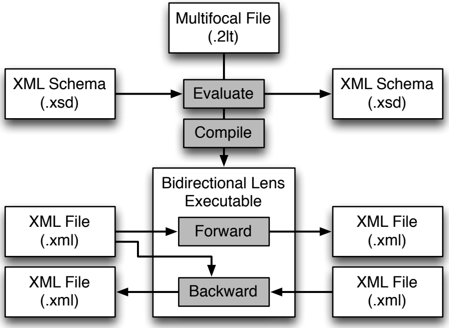


1. specialize the XPath expression ($/m/p$) for the source schema
2. convert it to a lens into the query's result type

Application Example: Multifocal XML Views

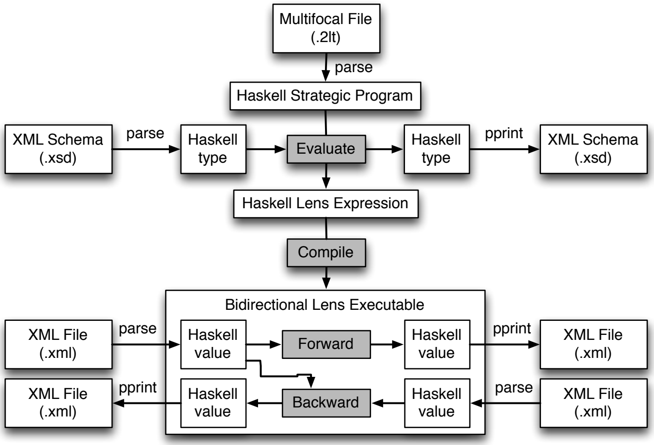
```
1   everywhere (try (at "series" erase))
2   >> everywhere (try (at "movie" (
     outermost (when "reviews" (
      select "count(//comment)" >> plunge "@popularity"))
     >> outermost (when "boxoffices" (
      select "sum(//@value)" >> plunge "@profit")))))
3   >> everywhere (try (at "actor" (
     outermost (at "played" (
      select "award/@name" >> all (rename "awname"))))))
```

Multifocal Framework



- two stages:
  1. evaluate: XML Schema $\Rightarrow$ XML Schema + lens
  2. compile: lens $\Rightarrow$ executable

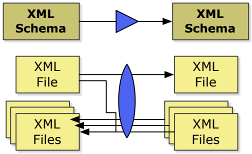
Implementation

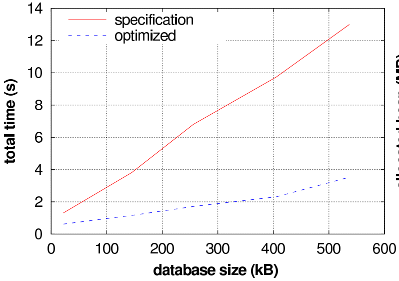


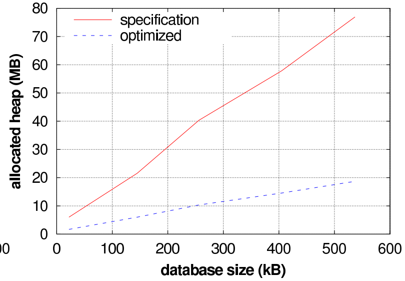
- XML-Haskell front-ends (type-safe)
- strategic two-level library (Haskell)
- bidirectional lens library (Haskell)

Optimization

- resulting lenses could be more efficient
- support for automatic optimization
- optimize lenses ⇒ generate efficient executables
- once-a-time penalty
- propagate multiple updates







Conclusions

+ Multifocal language for XML Schema evolution
+ strategic: concise and generic
+ two-level: multiple views, free document migrations
+ bidirectional: lenses, nice update semantics
+ staged optimization
+ framework (type-safe)

– expressiveness of the bidirectional language (views)
– alignment (parameterization)
– integration with XML technologies

Demo: Tool / Library

- more (recursive) XML Schema evolution examples
- `http://hackage.haskell.org/package/multifocal`