Bidirectional Data Transformation by Calculation

PhD Thesis Proposal

Hugo Pacheco

Supervisor: Alcino Cunha
Co-supervisor: José Nuno Oliveira

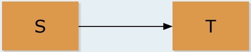
Departamento de Informática
Universidade do Minho, Braga, Portugal

I MAP-i Doctoral Symposium

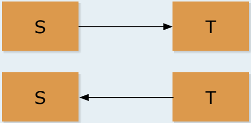
Braga - July 3th 2008

Data transformation

- frequent in software engineering
- essential to “bridge the gap” between the large offer on data formats

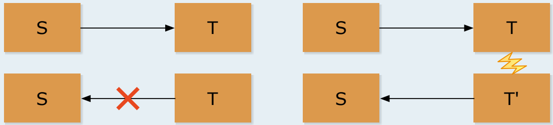


- many times we want to be able to transform in both directions
- 2 transformations

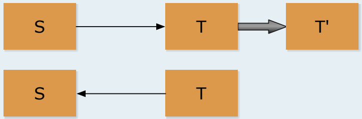


Bidirectionalizing data transformation

- expensive to write 2 transformations
- error-prone



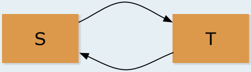
- likely to cause a maintenance problem



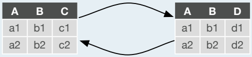
Bidirectional data transformation

Solution

- derive both from a single expression

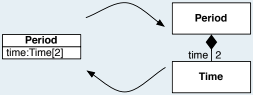


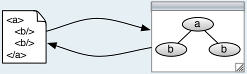
Bidirectional approaches exist for...



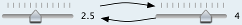
A	B	C
a1	b1	c1
a2	b2	c2

A	B	D
a1	b1	d1
a2	b2	d2



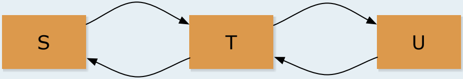


```
<a>
  <b/>
  <b/>
</a>
```



Bidirectional languages

- many informal approaches exist with unclear semantics
- give semantics to bidirectional transformations
- strong properties
- compositional approaches



- neat balance between expressiveness and robustness

Classification of bidirectional transformations

Refinements

$$\begin{tikzcd} A \arrow[r, bend left, "to"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "from"] \end{tikzcd}$$

Lenses

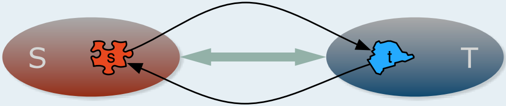
$$\begin{tikzcd} C \arrow[r, bend left, "get"] \arrow[r, phantom, "\geq"] & A \arrow[l, "create"'] \arrow[l, bend left=50, "put"] \end{tikzcd}$$

Isomorphisms

$$\begin{tikzcd} A \arrow[r, bend left, "iso"] \arrow[r, phantom, "\cong"] & B \arrow[l, bend left, "iso^\circ"] \end{tikzcd}$$

Two-level transformation

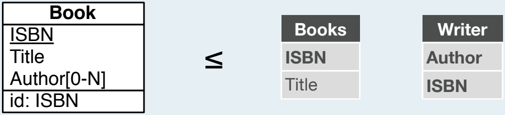
- type-level transformation of a data format
- value-level transformation of data instances
- a bidirectional two-level transformation:



- bound to the type system

Data refinement

- abstract specifications into low-level concrete implementations



Book
ISBN
Title
Author[0-N]
id: ISBN

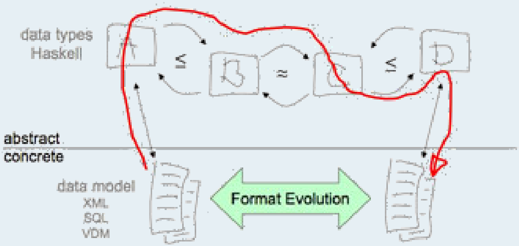
$\leq$

Books
ISBN
Title

Writer
Author
ISBN

2LT Framework

- two-level data refinement
- type-safe Haskell implementation
- universal representation of types



- strategic rewrite system
- visit `http://2lt.googlecode.com`

M A P i

Transforming the example

flatten nested map rule

$$A \rightharpoonup B \times (C \rightharpoonup D) \leq (A \rightharpoonup B) \times (A \times C \rightharpoonup D)$$

Book
ISBN
Title
Author[0-N]
id: ISBN

$$\begin{aligned} &\cong \quad \text{ISBN} \rightharpoonup \text{Title} \times [\text{Author}] \\ &\qquad \{ \text{ each Author is unique for each Book } \} \\ &\leq \quad \text{ISBN} \rightharpoonup \text{Title} \times (\text{Author} \rightharpoonup 1) \\ &\qquad \{ \text{ flatten nested map } \} \\ &\leq \quad (\text{ISBN} \rightharpoonup \text{Title}) \times (\text{ISBN} \times \text{Author} \rightharpoonup 1) \end{aligned}$$

$\cong$

Books
ISBN
Title

×

Writer
Author
ISBN

Type invariants

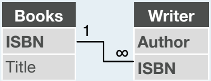
- preserve important structural information
- stricter transformation domains
- constraints on values
- stronger enough invariants lead to isomorphisms

flatten nested map rule (with invariant)

$$\mathsf{A} \rightharpoonup \mathsf{B} \times (\mathsf{C} \rightharpoonup \mathsf{D}) \cong (\mathsf{A} \rightharpoonup \mathsf{B}) \times (\mathsf{A} \times \mathsf{C} \rightharpoonup \mathsf{D})_{set \pi_1 \circ \delta \circ \pi_2 \subseteq \delta \circ \pi_1}$$

$$\begin{tikzcd} A \rightharpoonup B \times (C \rightharpoonup D) \cong (A & \rightharpoonup B) \times & (A \times C \rightharpoonup D) \arrow[ll, bend right, "fk"'] \end{tikzcd}$$

$$(\mathsf{ISBN} \rightharpoonup \mathsf{Title}) \times (\mathsf{ISBN} \times \mathsf{Author} \rightharpoonup 1)_{set \pi_1 \circ \delta \circ \pi_2 \subseteq \delta \circ \pi_1}$$



$\cong$

Calculation

- transformations are calculated through composition of smaller single-step transformations

$$\begin{tikzcd}
A \arrow[r, bend left, "to'"] \arrow[r, phantom, "\leq"] & B \arrow[l, bend left, "from'"] \arrow[r, bend left, "to''"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "from''"]
\end{tikzcd}$$

- point-free program calculation

$$\begin{tikzcd}
A \arrow[r, bend left, "to'' \bullet to'"] \arrow[r, phantom, "\leq"] & C \arrow[l, bend left, "from' \bullet from''"]
\end{tikzcd}$$

- simplification by rewriting

MAPi

Recursive types?

- currently only non-recursive types are supported
- hard to represent recursive types
- limited to the Haskell type system

We plan to study the support for ...

- single-recursive inductive types

  Alcino Cunha and Hugo Pacheco.
  Algebraic Specialization of Generic Functions for Recursive Types.
  Accepted to the 2nd workshop on Mathematically Structured Functional Programming, July 2008.

- mutually-inductive types
- nested types

Relations?

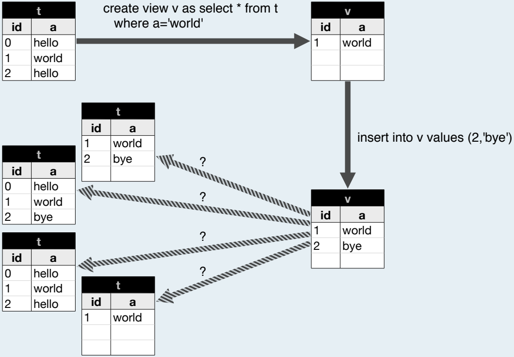
- although we want transformations to be functions

A calculus on relations may be beneficial to ...

- reverse transformations
  (Every relation R has a converse relation R^{-1})
- invariants as coreflexive relations
  (A relation R is coreflexive if $R \subseteq id$)
- deal with ambiguity - when one source schema has many correspondences in the target schema
  (Relations are composable)

View-update problem

- difficulty of choosing an unique database update for each view update



create view v as select * from t where a='world'

t

id	a
0	hello
1	world
2	hello

v

id	a
1	world

insert into v values (2,'bye')

v

id	a
1	world
2	bye

t

id	a
1	world
2	bye

t

id	a
0	hello
1	world
2	bye

t

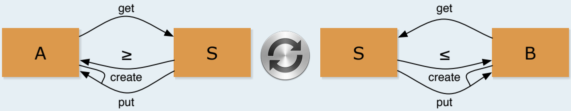
id	a
0	hello
1	world
2	hello

t

id	a
1	world

Harmony Framework

- two-level view-update (lenses)
- domain-specific languages
- data synchronization



- visit http://www.seas.uppenn.edu/~harmony

Lens languages

Very precise type systems for...

- unordered trees (sets of trees)
  - local tree transformations
  - conditionals
  - tree traversals
- relational databases (schemas with functional dependencies)
  - relational algebra (fusion, projection, selection)
- strings (regular expressions)
  - problems with ordered data ⇐ direct manipulation
  - positional alignment vs reorderable chunks with keys

Quotient lenses

- ignore inessential differences
- loosen lens properties
- equivalences on values
- the properties can be relaxed until they define isomorphisms

Examples

- ignore whitespaces
- reordering of attributes
- data duplication?

Conclusion

- similar to type invariants

Lens calculation?

- adapt lenses into the 2LT Framework

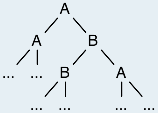
We want to investigate...

- type-safe representation of lenses (inherited)
- point-free calculus for lenses (inhereted)
- lenses for schema evolution and data mapping
- generic definition of lenses over recursive types
- strategic rewriting for lenses

Graph transformations?

- relevant for transformation of UML-like models
- representation of graphs in Haskell is problematic

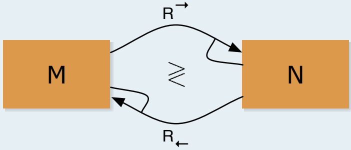
```
data A = A A B
data B = B B A
```



- related work:
  - GreAT (Graph Rewriting And Transformation language) (unidirectional)
  - VIATRA (VIsual Automated model TRAnsformations) (unidirectional)
  - AToM (A Tool for Multi-formalism and Meta-Modeling) (triple graph grammars - bidirectional)
  - BOTL (Bidirectional Object-oriented Transformation Language) (hybrid - bidirectional)

More general transformations?

- both the source and target can be modified and have state



- can add/delete information

Can they be created by composing refinements and lenses?

- $C \geq A \leq D \leq B \geq E$ (random composition?)
- $A \geq S \leq B$ (lens synchronization?)
- $A \cong B \cong C$ (restrict to isomorphisms?)

Schedule

	2009				*2010*				2011			
Other recursive types	▨	▨										
Calculus with relations and invariants			▨	▨								
Lenses à la 2LT					▨							
Lenses over recursive types					▨	▨						
Transformations for graph-like models							▨	▨				
Mixing refinements and lenses									▨	▨		
Thesis writing											▨	▨