Bidirectional Data Transformation by Calculation

Hugo Pacheco

Supervisor: Alcino Cunha
Co-supervisor: José Nuno Oliveira

DETI
Universidade de Aveiro, Portugal

III MAP-i Doctoral Symposium

Aveiro - July 6th 2010

Motivation

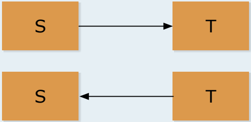
Unidirectional transformations

- Data transformations abound in software engineering

- Ideally, unidirectional transformations would suffice

Bidirectional transformations (classical approach)

- In real MDSE scenarios, we need to run a transformation backwards

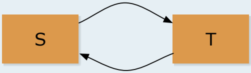


- Manual semantics
- Expensive, error-prone and a maintenance problem

Bidirectional languages

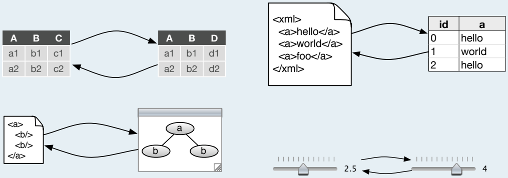
Bidirectional transformations (better approach)

- Derive both from the same specification



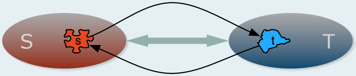
- Clean semantics
- Compositional

Bidirectional languages exist for...

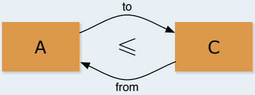


2LT Framework

- a bidirectional two-level transformation:

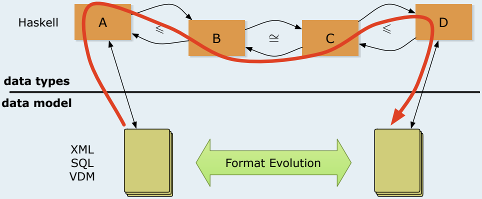


- a bidirectional two-level refinement:



$$from \circ to = id$$

- framework:



Calculation

- compositionality = cluttering

$$\begin{tikzcd}
A \arrow[r, bend left, "\mathrm{to}_1"] \arrow[r, phantom, "\leqslant"] & B \arrow[l, bend left, "\mathrm{from}_1"] \arrow[r, bend left, "\mathrm{to}_2"] \arrow[r, phantom, "\leqslant"] & C \arrow[l, bend left, "\mathrm{from}_2"]
\end{tikzcd}$$

$$\begin{tikzcd}
A \arrow[r, bend left, "\mathrm{to}_2 \cdot \mathrm{to}_1"] \arrow[r, phantom, "\leqslant"] & C \arrow[l, bend left, "\mathrm{from}_1 \cdot \mathrm{from}_2"]
\end{tikzcd}$$

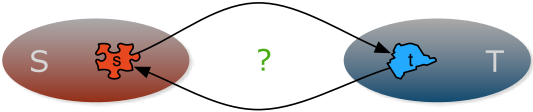
- point-free program calculation

$$\begin{aligned}
\circ &: (B \to C) \to (A \to B) \to (A \to C) \\
\pi_1 &: A \times B \to A \\
\triangle &: (A \to B) \to (A \to C) \to (A \to B \times C)
\end{aligned}$$

$$f \circ (g \circ h) = (f \circ g) \circ h \qquad \circ\text{-}\textsc{Assoc}$$

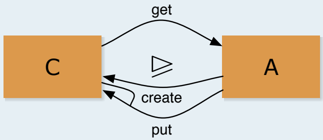
$$\pi_1 \circ (f \triangle g) = f \wedge \pi_2 \circ (f \triangle g) = g \qquad \times\text{-}\textsc{Cancel}$$

1 Extend the range of applications



Hugo Pacheco and Alcino Cunha
Generic Point-free Lenses.
Mathematics of Program Construction, 2010.

Lens



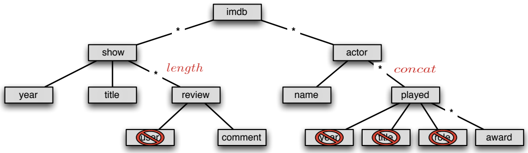
$$get \circ create = id$$
$$get \circ put = \pi_1$$
$$put \circ (get \bigtriangleup id) = id$$

Grammar for lens combinators

$$
\begin{array}{lcl}
Lens & ::= & id \mid Lens \circ Lens \mid !^c \mid Prod \mid Sum \mid Iso \mid Rec \\
Prod & ::= & {\pi_1}^b \mid {\pi_2}^a \mid Lens \times Lens \\
Sum & ::= & Lens \mathbin{{}_\bullet\nabla} Lens \mid Lens \mathbin{\nabla_\bullet} Lens \mid Lens + Lens \\
 & \mid & i_1 \nabla Lens \mid Lens \nabla i_2 \\
Iso & ::= & assocl \mid assocr \mid coassocl \mid coassocr \\
 & \mid & swap \mid coswap \mid distl \mid distr \\
Rec & ::= & in_F \mid out_F \mid F \cdot \mid (\![\cdot]\!)_F \mid [\!(\cdot)\!]_F
\end{array}
$$

Lens examples

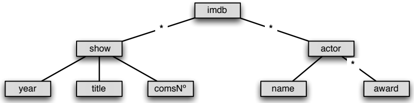
$$\begin{aligned}
length^a &= [\![(id + \pi_2{}^a) \circ out]\!] &: [A] \trianglerighteq \mathbb{N} \\
map\ f &= (\![in \circ (id + f \times id)]\!) &: [A] \trianglerighteq [A] \\
concat &= (\![\ldots]\!) &: [[A]] \trianglerighteq [A]
\end{aligned}$$



$$ex = map\ show \times map\ actor$$

$$show = id \times (id \times length \circ map\ (id \times \pi_{Comment}))$$

$$actor = id \times concat \circ map\ \pi_{Awards}$$



Lens calculation

- lift the point-free laws to lenses:

$$\pi_1{}^{a} \circ (f \times g) = f \circ \pi_1{}^{create_f\ a} \qquad \times\text{-}\textsc{Cancel}$$

$$(f \mathbin{\bullet\!\nabla} g) \circ (h + i) = f \circ h \mathbin{\bullet\!\nabla} g \circ i \qquad +\text{-}\textsc{Absor}$$

$$f \circ (\![g]\!)_F = (\![h]\!)_F \Leftarrow f \circ g = h \circ F\ f \qquad \textsc{Cata-Fusion}$$

- truly bidirectional calculus $\Leftarrow$ lens level
- Imdb example:

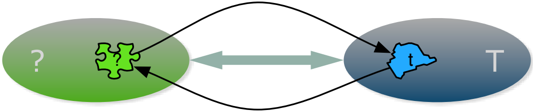
$$length^{a} \circ map\ f = length^{create_f\ a} \qquad \textsc{length-Map}$$

$$concat \circ map\ f = concatMap\ f \qquad \textsc{concat-Map}$$

- rewrite system for point-free lens simplification?

Alcino Cunha and Hugo Pacheco
Algebraic Specialization of Generic Functions for Recursive Types.
Mathematically Structured Functional Programming, 2008.

2 Extend the range of data formats



Alcino Cunha and Hugo Pacheco

Mapping between Alloy specifications and database implementations.

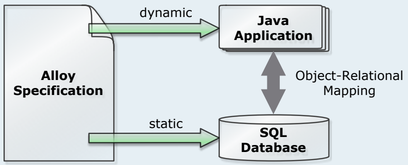
Software Engineering and Formal Methods, 2009.

AlloyDB

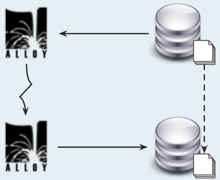
- bijective mapping between a subset of Alloy and relational schemas with functional and inclusion dependencies:

$$\begin{tikzcd} \text{Alloy}^{DB} \arrow[d, bend left, "Reflect"] \\ \text{Database} \arrow[u, bend left, "Reify"] \end{tikzcd}$$

Refine specifications into database-intensive applications



Reengineer legacy databases into specifications



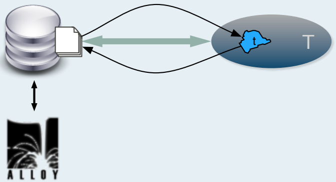
2LT integration?

- semi ad-hoc bidirectional transformation

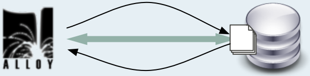
Question

How can we plugin this work into the 2LT framework?

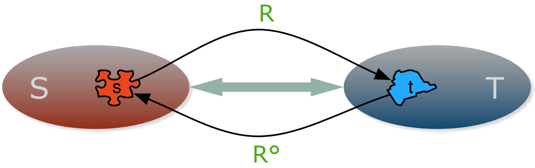
Alloy interface?



Fully bidirectional?



3 Investigate transformations as relations



Yet to come...