Bidirectionalization Transformation Based on Automatic Derivation of View Complement Functions

Kazutaka Matsuda*  
Zhenjiang Hu*, Keisuke Nakano*  
Makoto Hamana**, Masato Takeichi*

*The University of Tokyo  **Gunma University
Bidirectional Transformation

- “Bidirectional Transformation” = “View Function” + “Backward Transformation”

\[ f : S \rightarrow V \]
\[ r : (S \times V) \rightarrow S \]

[Hehner 90] and so on
Example: students

students [] = []
students (Std n g m:ms) = Std n g m:students ms
students (Prf n p m:ms) = students ms

students_B([],[]) = []
students_B(Std n g m:ms, Std n’ g’ m’:ss) = Std n’ g’ m’:students_B(ms,ss)
students_B(Prf n p m:ms, ss) = Prf n p m:students_B(ms,ss)
Example: students

students [] = []
students (Std n g m:ms) = Std n g m:students ms
students (Prf n p m:ms) = students ms

Problem

• Consistency of Both Programs
• Backward program for changed forward program?
Existing Work

• Bidirectional Combinators [Foster et al. 05]
  – 😊 High expressive power thanks to combinators
  – ☹️ Manually-defined basic transformations

• Constant Complement Bidirectionalization [Bancilhon & Spyratos 81]
  – 😊 Automatic generation of backward transformations
  – ☹️ No discussion of algorithms to derive complements
    • for RDB [Laurent et al. 01, Lechtenböger & Vossen 03]
Our Contribution
-- Combining Both Advantages --

- Bidirectionalization Transformation
  - for basic view functions
    - Affine and Treeless language
  - based on constant complement bidirectionalization
  - by 2 analyses and 3 program transformations
    - Range Analysis and Injectivity Analysis
    - Complement Derivation, Tupling and Inversion
Bidirectionalization
Based on Constant-Complement

\[ \langle f, g \rangle = \langle f, g \rangle^{-1}(v, g(s)) \]

**Complement Function**

- \( g \) is a complement function of \( f \) if \( \langle f, g \rangle \) is injective.

- \( \langle f, g \rangle(x) = (f(x), g(x)) \)

- \( g \) preserves lost information through transformation by \( f \)

**Backward Transformation**

- \( r(s, v) = \langle f, g \rangle^{-1}(v, g(s)) \)

[Bancilhon & Spyrou 81]
Backward Transformations using Complements

- A complement yields a backward transformation.

Complements of:

- \( \text{fst}(\text{Pair}(x,y)) = x \)
- \( \text{snd}(\text{Pair}(x,y)) = y \)
- \( \text{id}(\text{Pair}(x,y)) = \text{Pair}(x,y) \)

...more than one

- With \( \text{snd} \)...
- With \( \text{id} \)...

Good!

Bad!

(1,2)  \( \rightarrow \)  1  \( \rightarrow \)  2  \( \rightarrow \)  (4,2)

(1,2)  \( \rightarrow \)  1  \( \rightarrow \)  (1,2)

(4,2)  \( \leftarrow \)  4  \( \leftarrow \)  2  \( \leftarrow \)  (1,2)

(4,2)  \( \leftarrow \)  4  \( \leftarrow \)  (1,2)

with \( \text{snd} \)

with \( \text{id} \)
Our System Overview

- View Function
- Bidirectionalization Engine
- Complement Function Derivation
  - Tupling
  - Inversion

Main Result

- $r : (S \times V) \rightarrow S$
  - $r(s, v) = \langle f, g \rangle^{-1}(v, g(s))$

- $R: (S \times V) \rightarrow \text{Bool}$
  - $R(s, v) \iff (v, g(s)) \in \langle f, g \rangle(S)$
View Function

Affine and Treeless View Definition

students([]) = []
students(Std(n,g,m):ms) = Std(n,g,m):students(ms)
students(Prf(n,p,m):ms) = students(ms)

• Affine
  • Each variables is used at most once.

• Treeless [Wadler 90]
  • Arguments of function calls are only variables.
Complement Derivation

Students([[]]) = []
students(Std(n,g,m):ms) = Std(n,g,m):students(ms)
students(Prf(n,p,m):ms) = students(ms)

students^c([[]]) = B_1
students^c(Std(n,g,m):ms) = B_2(students^c(ms))
students^c(Prf(n,p,m):ms) = B_3(n,p,m,students^c(ms))

Keep Discarded Variables
Record Computation Paths

Two Analyses:
• Range Analysis
• Injectivity Analysis

Complement Derivation

Two Analyses:
• Range Analysis
• Injectivity Analysis
Tupling

\[\text{students} \triangle ([]) = ([], B_1)\]
\[\text{students} \triangle (\text{Std}(n,g,m):ms) = (\text{Std}(n,g,m):s, B_2(t))\]

where \((s,t) = \text{students} \triangle (ms)\)
\[\text{students} \triangle (\text{Prf}(n,p,m):ms) = (s, B_3(n,p,m,t))\]

where \((s,t) = \text{students} \triangle (ms)\)

Form of function

\[f_\Delta(p) = q\]
where \(x = g_\Delta(y)\)

\[f_\Delta(x) \equiv (f(x), f^c(x))\]
Inversion

\[ f_\triangle(p) = q \text{ where } x = g_\triangle(y) \]

\[ f_\triangle^{-1}(q) = p \text{ where } y = g_\triangle^{-1}(x) \]

Non-deterministic Program

Deterministic inverse for all the tested functions

students_\triangle^{-1}([], B_1) = []

students_\triangle^{-1}(Std(n,g,m):s, B_2(t)) = Std(n,g,m):ms

where ms = students_\triangle^{-1}(s,t)

students_\triangle^{-1}(s, B_3(n, p, m, t)) = Prf(n, p, m):ms

where ms = students_\triangle^{-1}(s,t)
Backward Transformation

Put them all together

\[ \text{students}_B(s,v) = \text{students}_\Delta^{-1}(v,\text{students}^c(s)) \]
View Update Checker

Checking if an updated view can be reflected

Suppose an initial source is $P_1 P_2 S_1 P_3 S_2$

Bottom-up tree automaton

It prohibits invalid view updates.

- $S_1 S_2$
- $S_1 S_2 S_3$

Std($q_{\text{any}}, q_{\text{any}}, q_{\text{any}}) : q_1 \rightarrow q_{\text{accept}}$

Std($q_{\text{any}}, q_{\text{any}}, q_{\text{any}}) : q_2 \rightarrow q_1$

$[] \rightarrow q_2$
Conclusion

• Bidirectionalization transformation
  – based on derivation of complement functions
  – for affine and treeless programs
• View update checker
• System implementation
  – [http://www.ipl.t.u-tokyo.ac.jp/~kztk/bidirectionalization/](http://www.ipl.t.u-tokyo.ac.jp/~kztk/bidirectionalization/)
  – Functions that we have tested in the system
    • fst, snd, half, add, max, min, zip, append, map, filter
Future Work

• Widening class of language
  – Duplications and function compositions

• Applying derivation principles to practical examples
  – XML transformations