An axiomatic basis for bidirectional programming

Josh H-S Ko\textsuperscript{1} and Zhenjiang Hu\textsuperscript{1,2}

\textsuperscript{1} National Institute of Informatics (NII), Japan
\textsuperscript{2} SOKENDAI (The Graduate University for Advanced Studies), Japan

Symposium on Principles of Programming Languages (POPL)
11 January 2018, Los Angeles, CA, US
Lenses
(asymmetric & state-based)
Extraction & update

(source) 

(a, b, c)  

(view)  

get b  

put b' 

(a, b', c)
Well-behaved lenses

\[
\text{get} : S \to V \\
\text{put} : S \to V \to S
\]

\[
\text{get} (\text{put} \ s \ v) = v \\
\text{put} \ s \ (\text{get} \ s) = s
\]
Get

map f \circ filter p

0

1

2
Put
Put

1 2 2 0 4
Put

2
0
4
1
Put

2
0
4

2
0
4
Bidirectional programming languages
one program for two directions
Bidirectional programming

write one program to control two directions
“Get-based” approach

- `map f <alignment strategy>`
- `filter p <management of ignored elements>`
“Put-based” approach

align p match b create conceal =
  case
    normal [] [] exit []
    rearrV [] -> ()
    skip const ()
    normal (s::_) (v::_) | p s && match s v exit (s::_) | p s
    rearrS (s::ss) -> (s, ss)
    rearrV (v::vs) -> (v, vs)
    b * align p match b create conceal
  adaptive (s::_) [] | p s
  
  ss = (prefix, remaining) = span s p

If put is well-behaved with both get and get’,
then get = get’.

bidirectional programming ➔ unidirectional programming
You still need to figure out the get behaviour!

Yes, but we can do it just from the put direction...
Get is really a part of put.
Synchronisation

Maintaining a consistency relation

get : S → V
“executable” consistency relation

put : S → V → S
consistency restorer

get (put s v) = v
correctness

put s (get s) = s
hippocraticness
“Get-based” approach

First: write a consistency relation

map f <alignment strategy>
  filter p <management of ignored elements>

Second: annotate the consistency relation with restoration behaviour
“Put-based” approach

\[
\text{align } p \text{ match } b \text{ create } \text{conceal} = \\
\text{case} \\
\quad \text{normal } [] [] \text{ exit } [] \\
\quad \text{rarrV } [] \rightarrow () \\
\quad \text{skip const } () \\
\quad \text{normal } (s::) (v::) | \ p \ s \ &\& \ \text{match } s \ v \ \text{exit } (s::) | \ p \ s \\
\quad \text{rarrS } (s::ss) \rightarrow (s, ss) \\
\quad \text{rarrV } (v::vs) \rightarrow (v, vs) \\
\quad b * \ \text{align } p \ \text{match } b \ \text{create } \text{conceal} \\
\quad \text{adaptive } (s::) [] | \ p \ s \\
\quad \quad \text{let } \text{prefix, remaining} = \text{span } p \ ss \\
\quad \quad \text{map } \text{conceal } \text{prefix} \ \text{++} \ \text{remaining} \\
\quad \text{normal } (s::) [] | \ not \ (p \ s) \ \text{exit } (s::) | \ not \ (p \ s) \\
\quad \text{rarrS } (s::ss) \rightarrow (s, ss) \\
\quad \text{rarrV } vs \rightarrow ((), vs) \\
\quad \text{skip const } () \\
\quad \text{align } p \ \text{match } b \ \text{create } \text{conceal} \\
\quad \text{adaptive } s (v::) | \ isJust \ (\text{findFirstMatch } v \ ss) \\
\quad \quad \text{ss } (v::) \rightarrow \text{uncurry } (::) \ (\text{fromJust } (\text{findFirstMatch } v \ ss)) \\
\quad \quad \text{adaptive } s (v::) | \ p \ (\text{create } v) \\
\quad \quad \quad \text{ss } (v::) \rightarrow \text{create } v \ :: \ ss \\
\]

First: write a program to restore a consistency relation in mind

Second: the consistency relation becomes executable for free
Put

0 --|---|--|---
     | 1 | 2 |

---|--|---|--|---
     | 2 | 0 | 4 |
map \ f \circ \ filter \ p
To program a consistency restorer, the programmer must have a consistency relation in mind.

A put-based language makes executable...
Formalise!
in terms of a program logic
BiGUL
Bidirectional Generic Update Language

lens combinators

rearrV v -> (v, ())
  replace * skip const ()

atomic lenses
An Axiomatic Basis for Bidirectional Programming

4.2 Product

Given two

∀

and

i

←

adaptive

R

s

∗


∀

and

i

←

adaptive

R

s

∗

Figure 2
Reasoning

\[
\begin{align*}
\{ \_ \_ \_ \} \\
\text{rearrV} \; v & \rightarrow (v, ()) \\
\{ \_ (\_, ())) \} \\
\{ \_ \_ \_ \} \\
\text{replace} \\
\{ \; w' \_ v \mid w' = v \; \} \\
\ast \{ \; \_ () \; \} \\
\{ \; \_ () \mid \text{const} () \; s = () \; \} \\
\text{skip} \; \text{const} () \\
\{ \; h' \; h () \mid h' = h \; \} \\
\{ \; (w', h') (\_, h) (v, ()) \mid w' = v \land h' = h \; \} \\
\{ \; (w', h') (\_, h) v \mid w' = v \land h' = h \; \}
\end{align*}
\]
Main theorem

If \( \{ s \mid v \in R s v \} \subseteq b \{ s' \mid v \in C s' v \} \)
then \( b \cdot \text{get} \cap R \subseteq C \)
A part of get behaviour can be found in put triples.
Main theorem

If \( \{ s \mid R \subseteq C \} \) then \( \{ s \mid C \subseteq C \} \)

\[
\begin{align*}
\text{If} & \quad \{ s \mid R \subseteq C \} \\
\text{then} & \quad b.\text{get} \cap R \subseteq C
\end{align*}
\]

\[
\begin{align*}
\{ s \mid \text{False} \} & \quad b \quad \{ s' \mid C \subseteq C \} \\
\{ s \mid \text{True} \} & \quad b \quad \{ s' \mid C \subseteq C \}
\end{align*}
\]
Domain of get
Range of put
An Axiomatic Basis for Bidirectional Programming

\[
\{\{ s \mid R \} s v \}\ b \{\{ s' \mid P' \} s'\}
\]

Range triples

\[
\begin{align*}
\{\{ \emptyset \} & \fail \{\{ \emptyset \} \} & \{\{ s \mid s = v \} & \replace \{\_\} \} & \{\{ s \mid f \ s = v \} & \skip f \{\_\} \} \\
\{\{ L \} l & \{\{ P' \} r \} \{\{ R \} Q' \} \} & \{\{ L \} \times R \} l & r \{\{ P' \} \times Q' \} \} & \{\{ L \} \times R \} l & r \{\{ P' \} \times Q' \} \} \} \} \}
\end{align*}
\]

where

\[
P' = \{\{ P' \prime \} | \ r = \text{(normal M exit E \mathbin{\|} b)} \in bs \}
\]
Main theorem MK II

If \( \{ s \mid \mathcal{R} s \} \uparrow b \{ \{ s' \mid \mathcal{C} s' \} \uparrow \}
and \{\{ s \mid \mathcal{R} s \} \} \uparrow b\{\{ s' \mid \mathcal{P}' s' \}\}
then \( b \).get is defined on \( \mathcal{P}' \)
and \( b \).get\mid\( \mathcal{P}' \) \( \subseteq \) \( \mathcal{C} \)
Get behaviour can be found in put and range triples.
Also in the paper

• An introduction to BiGUL in terms of the axiomatic semantics

• Recursion rules and key-based alignment

• Everything formalised in Agda
Get is really a part of put.

Get behaviour can be found in put and range triples.

bidirectional programming ➔ unidirectional programming
An axiomatic basis for bidirectional programming

Josh H-S Ko and Zhenjiang Hu (NII & SOKENDAI, Japan)

get : S → V
put : S → V → S
get (put s v) = v
put s (get s) = s

unidirectional programming
one program for two directions

{ R } b { R' }
{ { R } } b { { P' } }