An axiomatic basis for bidirectional programming

Josh H-S Ko\(^1\) and Zhenjiang Hu\(^{1-3}\)

\(^1\) National Institute of Informatics (NII), Japan
\(^2\) University of Tokyo, Japan
\(^3\) SOKENDAI (The Graduate University for Advanced Studies), Japan

International Workshop on Bidirectional Transformations (BX)
10 April 2018, Nice, France
Towards a general-purpose bidirectional language

Josh H-S Ko\textsuperscript{1} and Zhenjiang Hu\textsuperscript{1-3}

\textsuperscript{1} National Institute of Informatics (NII), Japan
\textsuperscript{2} University of Tokyo, Japan
\textsuperscript{3} SOKENDAI (The Graduate University for Advanced Studies), Japan

International Workshop on Bidirectional Transformations (BX)
10 April 2018, Nice, France
BiGUL
Bidirectional Generic Update Language

lens combinators

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

atomic lenses
replace.put s v = v
Hoare-style logic

\[
\begin{align*}
\{ s \, v \mid \text{True} \} & \quad \textbf{replace} \quad \{ s' \, s \, v \mid s' = v \}
\end{align*}
\]

An Axiomatic Basis for Bidirectional Programming

\[
\begin{align*}
\{ \emptyset \} & \quad \text{fail} \quad \{ \emptyset \} \\
\{ \_ \_ \} & \quad \text{replace} \quad \{ s' - v \mid s' = v \} \\
\{ s \, v \mid f \, s = v \} & \quad \text{skip} \quad \{ s' \, s' \mid s' = s \}
\end{align*}
\]

\[
\begin{align*}
\{ L \} & \quad l \{ L' \} \quad \{ R \} \quad r \{ R' \} \\
\{ L \times R \} & \quad l \times r \{ L' \times R' \} \\
T & \subseteq R \{ R \} \quad b \{ R' \} \quad R' \cap \{ s \, v \} \quad T \, s \, v \} \subseteq T'
\end{align*}
\]

\[
\begin{align*}
\{ s \, \text{wpat} \mid R \, \text{wpat} \} & \quad b \{ s' \, s \, \text{wpat} \mid R' \, s' \, \text{wpat} \} \\
\{ s \, \text{vpat} \mid R \, \text{vpat} \} & \quad \text{rearrV \, vpat} \rightarrow \text{vpat} \parallel b \{ s' \, s \, \text{vpat} \mid R' \, s' \, \text{vpat} \} \\
\{ tpat \, v \mid R \, \text{tpat} \, v \} & \quad b \{ tpat' \, tpat \, v \mid R' \, \text{tpat'} \, \text{tpat} \, v \} \\
\{ \text{spat} \, v \mid R \, \text{spat} \, v \} & \quad \text{rearrS \, spat} \rightarrow \text{tpat} \parallel b \{ \text{spat'} \, \text{spat} \, v \mid R' \, \text{spat'} \, \text{spat} \, v \}
\end{align*}
\]

\[
\forall(\text{normal} \, M \, \text{exit} \, E \downarrow b) \in bs.
\]

\[
\begin{align*}
\{ R \cap \widetilde{M} \} & \quad b \{ R' \cap \{ s' \_ - v \mid \widetilde{M} \, s' \, v \wedge \widetilde{E} \, s' \} \}
\end{align*}
\]

\[
\forall(\text{adaptive} \, M \downarrow f) \in bs.
\]
Reasoning

{ _ _ }

rearrV \ v \rightarrow (v, ())
{ _ (_ , ())) }
{ _ _ }

replace

{ w' _ v | w' = v }

* { _ () }
{ _ () | const () s = () }

skip const()
{ h' h () | h' = h }

{ (w', h') (_, h) (v, ()) | w' = v \land h' = h }
{ (w', h') (_, h) v | w' = v \land h' = h }
Putback triples

**precondition:** a binary relation on the original source and view

\[
\{ s \, v \mid R \, s \, v \} \quad b \quad \{ s' \, s \, v \mid R' \, s' \, s \, v \}
\]

**postcondition:** a ternary relation on the updated source, original source, and view

\[
\forall s, v. \quad R \, s \, v \quad \Rightarrow \quad \exists s'. \quad b.\text{put} \ s \ v = s' \\
\wedge \quad R' \, s' \, s \, v
\]

**soundness**
Get behaviour

If \{ s \mid R s v \} \quad b \quad \{ s' \mid C s' v \}

then \quad b.\text{get} \cap R \subseteq C

Proof

\begin{align*}
\text{b.get} & \quad s = v \\
\text{soundness} & \quad \text{b.put} \quad s \quad v = s' \\
\text{GetPut} & \quad s' = s \\
\text{C} & \quad s' \quad v \\
\text{C} & \quad s \quad v
\end{align*}
Range triples

**input range:** a binary relation on the original source and view

\[
\begin{array}{|c|c|}
\hline
s & v \\
R & s & v \\
\hline
\end{array}
\]  
\begin{array}{|c|c|}
\hline
b & s' \\
\hline
\end{array}
\begin{array}{|c|c|}
\hline
P' & s' \\
\hline
\end{array}

**output range:** a unary relation/predicate on the updated source

\[
\forall s'. \ P' s' \Rightarrow \exists v. \ b.\text{get} s = v \land R s v
\]

**soundness**

\[
\forall s'. \ P' s' \Rightarrow \exists s, v. \ b.\text{put} s v = s' \land R s v
\]
Main theorem MK II

If \( \{ s \mid R s v \} \) b \( \{ s' \mid C s' v \} \)
and \( \{\{ s \mid R s v \}\} \) b \( \{\{ s' \mid P' s' \}\} \)
then b.get is defined on \( P' \)
and \( b.get|P' \subseteq C \)
Key-based list alignment

```haskell
keyAlign ks kv b c =
    case
        normal [] [] exit []
        rearrV [] -> ()
        skip const ()
    normal (s::_) (v::_) | ks s == kv v exit (_::_)
        rearrS (s::ss) -> (s, ss)
        rearrV (v::vs) -> (v, vs)
        b * keyAlign ks kv b c
        adaptive (_::_) []
        \_ _ -> []
    adaptive ss (v::_) | kv v `elem` map ks ss
        \ss (v::_) -> extract ks kv v ss
        adaptive _ (_::_)
        \ss (v::_) -> c v :: ss
```
Verifying keyAlign

\{ ss \text{ vs } | \text{ True } \} 

\textbf{keyAlign} \ldots

\{ ss' ss \text{ vs } | \exists ss^*. \text{ ss* retains elements in ss and ss’ is an updated version of ss* and element-wise consistent with vs} \}

\{\{ ss \text{ vs } | \text{ True } \}\} \quad \textbf{keyAlign} \ldots \quad \{\{ ss' | \text{ True } \}\}

\{\ldots\} \ldots \{\ldots\}

\{\{\ldots\}\} \ldots \{\{\ldots\}\}
Evolution of session types

program/process

\[
x := \text{read ch} \\
\text{write ch ...} \\
\text{write ch ...}
\]

type/protocol

\[
\text{ch: ?int;} \\
\text{!int;} \\
\text{!(int \times int); end}
\]
Process–protocol synchroniser

- The get direction is type inference.
- For the put direction:
  - Retain the original process behaviour (assuming that the protocol is only being refactored or optimised)
  - Reject an update if a new protocol deviates too much from the original one
- Verification desirable
Towards a general-purpose bidirectional language
General-purpose bidir. lang.

- Synchronisation problems are ubiquitous and diverse.
  - Inventing a DSL for every problem?
- Reuse (and unification) of general BX concepts
  - “BenchmarX reloaded” at BX ’17
- Tony at SSBX ’16: Implementing TGG in BiGUL?
- Tool support — IDE, verifier, debugger, etc
What does a general-purpose bidirectional language look like?

For state-based asymmetric lenses...
“Get-based” approach

First: write a consistency relation (get)

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

Second: annotate the consistency relation with restoration (put) behaviour
“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
    adaptive (s::ss) [[] | p s
    adaptive (_v::_) [v | p

**First:** write a (put) program to restore a consistency relation in mind

**Second:** the consistency relation (get) becomes executable for free
Conclusion

- Declarative approaches (DSLs) and investigation into various forms of well-behavedness laws/principles are definitely useful.

- But general-purpose bidirectional languages should be given some thoughts too.
  - In addition to well-behavedness guarantees...
  - Max freedom to program and reason about the consistency restoration behaviour