当一
千入
Programming
Research
Laboratory

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

Programming
Research
Laboratory

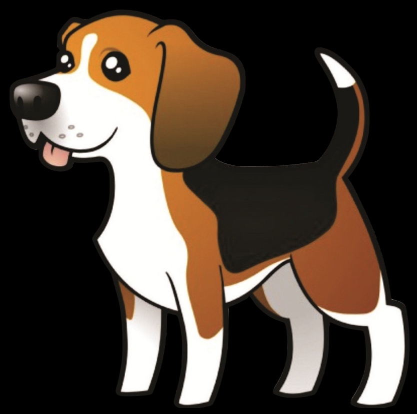
Towards a general-purpose bidirectional language

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

BiGUL



Bidirectional Generic Update Language

lens combinators

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

atomic lenses

```
replace.put s v = v
```

Hoare-style logic

$$\frac{}{\{\ s\ v \mid \text{True}\ \}\ \ \textbf{replace}\ \ \{\ s'\ s\ v \mid s' = v\ \}}$$

$$\frac{}{\{\,\emptyset\,\}\ \ \textbf{fail}\ \ \{\,\emptyset\,\}} \qquad \frac{}{\{\,\_\ \_\,\}\ \ \textbf{replace}\ \ \{\,s'\ \_\ v \mid s' = v\,\}} \qquad \frac{}{\{\,s\ v \mid f\ s = v\,\}\ \ \textbf{skip}\ f\ \ \{\,s'\ s\ \_ \mid s' = s\,\}}$$

$$\frac{\{\,L\,\}\ \ l\ \ \{\,L'\,\} \qquad \{\,R\,\}\ \ r\ \ \{\,R'\,\}}{\{\,L * R\,\}\ \ l * r\ \ \{\,L' * R'\,\}} \qquad \frac{T \subseteq R \qquad \{\,R\,\}\ \ b\ \ \{\,R'\,\} \qquad R' \cap \langle\,\_\ s\ v \mid T\ s\ v\,\rangle \subseteq T'}{\{\,T\,\}\ \ b\ \ \{\,T'\,\}}$$

$$\frac{\{\,s\ \mathit{wpat} \mid R\ s\ \overline{\mathit{wpat}}\,\}\ \ b\ \ \{\,s'\ s\ \mathit{wpat} \mid R'\ s'\ s\ \overline{\mathit{wpat}}\,\}}{\{\,s\ \mathit{vpat} \mid R\ s\ \overline{\mathit{vpat}}\,\}\ \ \textbf{rearrV}\ \mathit{vpat} \rightarrow \mathit{wpat} \Lsh b\ \ \{\,s'\ s\ \mathit{vpat} \mid R'\ s'\ s\ \overline{\mathit{vpat}}\,\}}$$

$$\frac{\{\,\mathit{tpat}\ v \mid R\ \overline{\mathit{tpat}}\ v\,\}\ \ b\ \ \{\,\mathit{tpat}'\ \mathit{tpat}\ v \mid R'\ \overline{\mathit{tpat}'}\ \overline{\mathit{tpat}}\ v\,\}}{\{\,\mathit{spat}\ v \mid R\ \overline{\mathit{spat}}\ v\,\}\ \ \textbf{rearrS}\ \mathit{spat} \rightarrow \mathit{tpat} \Lsh b\ \ \{\,\mathit{spat}'\ \mathit{spat}\ v \mid R'\ \overline{\mathit{spat}'}\ \overline{\mathit{spat}}\ v\,\}}$$

$\forall(\textbf{normal}\ M\ \textbf{exit}\ E \Lsh b) \in bs.$

$\{\,R \cap \widehat{M}\,\}\ \ b\ \ \{\,R' \cap \langle\,s'\ \_\ v \mid \widehat{M}\ s'\ v \wedge \widehat{E}\ s'\,\rangle\,\}$

$\forall(\textbf{adaptive}\ M \Lsh f) \in bs.$

Reasoning

```
{ _ _ }
rearrV v → (v, ())
  { _ (_, ()) }
    { _ _ }
    replace
    { w′ _ v | w′ = v }
  * { _ () }
    { _ () | const () s = () }
    skip const ()
    { h′ h () | h′ = h }
  { (w′, h′) (_, h) (v, ()) | w′ = v ∧ h′ = h }
{ (w′, h′) (_, h) v | w′ = v ∧ h′ = h }
```

Putback triples

precondition: a binary relation
on the original source and view

```
{ s v | R s v }  b  { s′ s v | R′ s′ s v }
```

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

soundness

```
∀s,v.  R s v  ⇒  ∃s′.  b.put s v = s′
                     ∧  R′ s′ s v
```

Get behaviour

If { s v | R s v } b { s′ _ v | C s′ v }

then b.get ∩ R ⊆ C

Proof

```
b.get s = v      soundness              GetPut
R s v            b.put s v = s′         s′ = s
                 C s′ v                 C s v
                                            ∎
```

Range triples

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

$$\{\{\ s\ v\ |\ R\ s\ v\ \}\}\ \ b\ \ \{\{\ s'\ |\ P'\ s'\ \}\}$$

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

soundness

$$\forall s'.\ \ P'\ s'\ \Rightarrow\ \exists v.\ \ b.get\ s = v \land R\ s\ v$$

$$\forall s'.\ \ P'\ s'\ \Rightarrow\ \exists s,v.\ \ b.put\ s\ v = s' \land R\ s\ v$$

Main theorem MK II

```
If     { s v | R s v }  b  { s’ _ v | C s’ v }
and   {{ s v | R s v }} b {{ s’ | P’ s’ }}

then  b.get is defined on P’
and   b.get|P’  ⊆  C
```

Key-based list alignment

```
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 vs | True }
keyAlign …
{ ss′ ss vs | ∃ss*. ss* retains elements in ss and
                ss′ is an updated version of ss* and
                    element-wise consistent with vs }

{{ ss vs | True }}  keyAlign …  {{ ss′ | True }}
```

```
 {…}   …   {…}
{{…}} … {{…}}
```

Evolution of session types

program/process

```
x := read ch

write ch …

write ch …
```

:

type/protocol

```
ch: ?int;
    !int;
    !int; end

ch: ?int;
    !(int × 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
```

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