in Agda!

BiGUL

A formally verified core language for putback-based bidirectional programming

Josh Ko [b], Tao Zan [a, b], and Zhenjiang Hu [a, b]

[a] SOKENDAI (The Graduate University for Advanced Studies), Japan
[b] National Institute of Informatics, Japan

Workshop on Partial Evaluation and Program Manipulation
19 Jan 2016, St Petersburg, FL, US

Bidirectional transformations (asymmetric lens version)

Source | **View**

POPL 2016
The annual Symposium on Principles of Programming Languages is a forum ...

PEPM 2016
The PEPM Symposium/Workshop series aims at bringing together researchers ...

get : S → V

POPL 2016
PEPM 2016

PEPM ’16
The PEPM Symposium/Workshop series aims at bringing together researchers ...

POPL ’16
The annual Symposium on Principles of Programming Languages is a forum ...

put : S → V → S

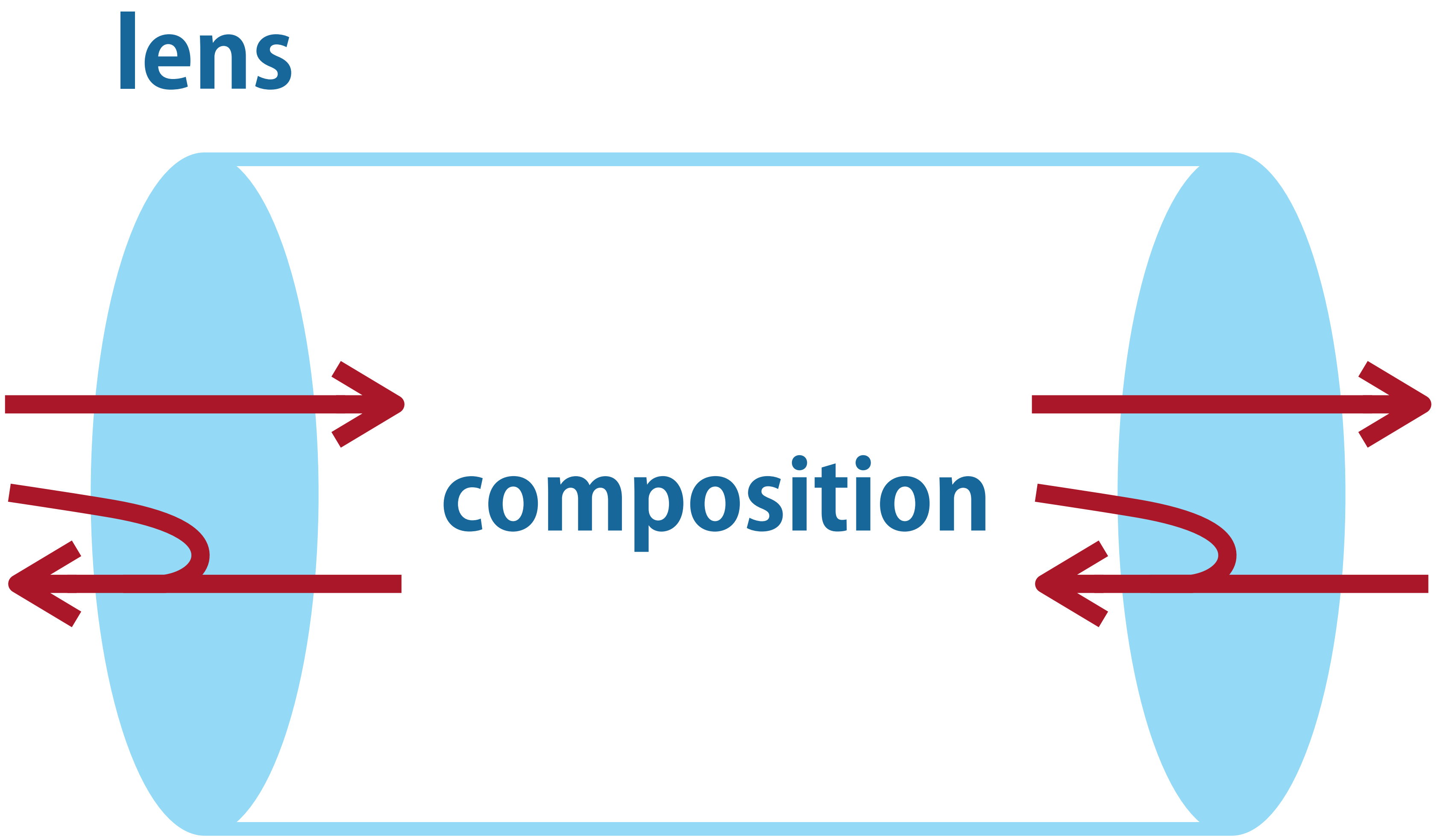
PEPM ’16
POPL ’16

Well-behavedness

PutGet :
get (put s v) ≡ v

GetPut :
put s (get s) ≡ s

Bidirectional programming with lenses (Foster et al., POPL ’05)



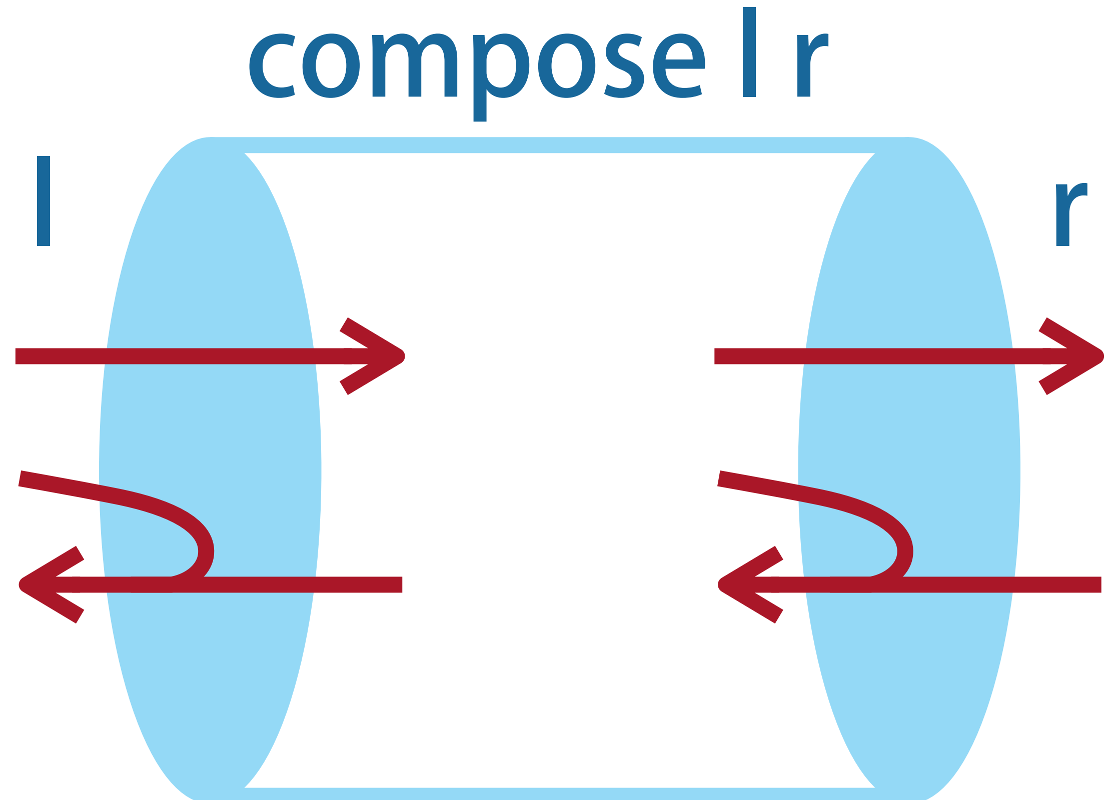
A trick for proving partial well-behavedness

Partial lenses

```
record Lens (S V : Set) : Set where
  field
    get    : S → Maybe V
    put    : S → V → Maybe S
    PutGet : put s v ≡ just s′  →  get s′ ≡ just v
    GetPut : get s ≡ just v  →  put s v ≡ just s

_>>=_ : Maybe A → (A → Maybe B) → Maybe B
```

Lens composition



compose : Lens A B → Lens B C → Lens A C
compose l r = record
  { get = λ a → l.get a ≫= λ b → r.get b
  ; put = λ a c → l.get a ≫= λ b → r.put b c ≫= λ b′ → l.put a b′
  ; PutGet = ? ; GetPut = ? }

Direct proof of PutGet

PutGet :

(l.get a ≫= λ b → r.put b c ≫= λ b′ → l.put a b′) ≡ just a′
  → (l.get a′ ≫= λ b → r.get b) ≡ just c

lemma :

(mx ≫= f) ≡ just y → ∃[x] (mx ≡ just x) × (f x ≡ just y)

```
PutGet p with lemma p
PutGet _ | (b, g, p) with lemma p
PutGet _ | (b, g, _) | (b′, p, q) rewrite l.PutGet q = r.PutGet p
```

Instead of decomposing proofs,
make the proofs decompose by themselves!

Deep embedding for defining two interpretations

```
data Par : Set → Set₁ where
  return : A → Par A
  _≫=_   : Par A → (A → Par B) → Par B

runPar : Par A → Maybe A
runPar (return x)  =  just x
runPar (mx ≫= f)  =  runPar mx ≫= (runPar ∘ f)

_↦_  :  Par A → A → Set
(return x) ↦ y  =  x ≡ y
(mx ≫= f) ↦ y  =  ∃[ x ]  (mx ↦ x) × (f x ↦ y)

px ↦ x   ↔   runPar px ≡ just x
```

Partial lenses

```
record Lens (S V : Set) : Set₁ where
  field
    get    : S → Par V
    put    : S → V → Par S
    PutGet : put s v ↦ s′  →  get s′ ↦ v
    GetPut : get s ↦ v  →  put s v ↦ s
```

Well-behavedness proofs become elementary programs!

PutGet :

$$(\mathrm{l.get\ a} \ggg\!= \lambda\, \mathrm{b} \to \mathrm{r.put\ b\ c} \ggg\!= \lambda\, \mathrm{b'} \to \mathrm{l.put\ a\ b'}) \mapsto \mathrm{a'}$$
$$\to \quad (\mathrm{l.get\ a'} \ggg\!= \lambda\, \mathrm{b} \to \mathrm{r.get\ b}) \mapsto \mathrm{c}$$

$$\|$$

$$\exists[\,\mathrm{b}\,]\ \ (\mathrm{l.get\ a} \mapsto \mathrm{b}) \times (\exists[\,\mathrm{b'}\,]\ \ (\mathrm{r.put\ b\ c} \mapsto \mathrm{b'}) \times (\mathrm{l.put\ a\ b'} \mapsto \mathrm{a'}))$$
$$\to \quad \exists[\,\mathrm{b}\,]\ \ (\mathrm{l.get\ a'} \mapsto \mathrm{b}) \times (\mathrm{r.get\ b} \mapsto \mathrm{c})$$

PutGet (b, g, b′, p, q) = (b′, l.PutGet q, r.PutGet p)

BiGUL as reported in the paper

Basic lenses

Source decomposition

View rearrangement

Case analysis on source

Case analysis on view

List alignment

The latest version of BiGUL

Basic lenses

Standard lens combinators

Source/view rearrangement

General case analysis (on both source and view)

List alignment ⇐ general case analysis + recursion

Haskell

A sample BiGUL$_{\text{Haskell}}$ program

```
updateSelected ::
  (s -> Bool) -> BiGUL s v -> (v -> s) -> BiGUL [s] [v]
updateSelected p b c = Case
  [ $(normalSV [p| [] |] [p| [] |])$
      $(rearrV [| \[] -> () |])$ Skip
  , $(adaptiveSV [p| [] |] [p| _:_ |])$
      \_ vs -> map c vs
  , $(normalSV [p| (p -> True):_ |] [p| _:_ |])$
      $(rearrS [| \(s:ss) -> (s, ss) |])$
        $(rearrV [| \(v:vs) -> (v, vs) |])$
          b `Prod` updateSelected p b c
  , $(adaptiveSV [p| (p -> True):_ |] [p| [] |])$
      \ss _ -> dropWhile p ss
  , $(normalS [p| (p -> False):_ |])$
      $(rearrS [| \(s:ss) -> ss |])$ updateSelected p b c
  ]
```

Issues we are trying to tackle

Totality

BiGUL programs are only guaranteed to be partially well-behaved — they can still fail inadvertently due to implicit dynamic checks.

Dependently typed lenses?

Functional correctness

Sometimes it is not easy to get BiGUL programs to work as intended (especially in the presence of dynamic checks and recursion).

Reasoning principles/tools needed

http://www.prg.nii.ac.jp/bx

Thanks!

What have been built on top of BiGUL

View-updating for relational databases

expressing more flexible view-updating strategies with a putback-based language

Parsing & “reflective” printing

describing a consistent pair of parser and “reflective” printer in a single program

Synchronisation of web server configuration files

unifying different configuration file formats to simplify the self-adaptation logic