

Bidirectionalizing Programs with Duplication through Complement Function Derivation

Kazutaka Matsuda
The University of Tokyo

joint work with
Zhenjiang Hu, Keisuke Nakano, Makoto Hamana, and Masato Takeichi

2008-11-17 GRACE Bx Workshop

Motivation

- How to derive "Side Effect"-free " f_B " from " f "?
 - Side-Effect free: no change on data that are irrelevant to view construction.
 - $S \sim V \times U$ & f_B keeps "U"-part equal
 - Constant Complement! [Bancilhon&Spyratos 81]

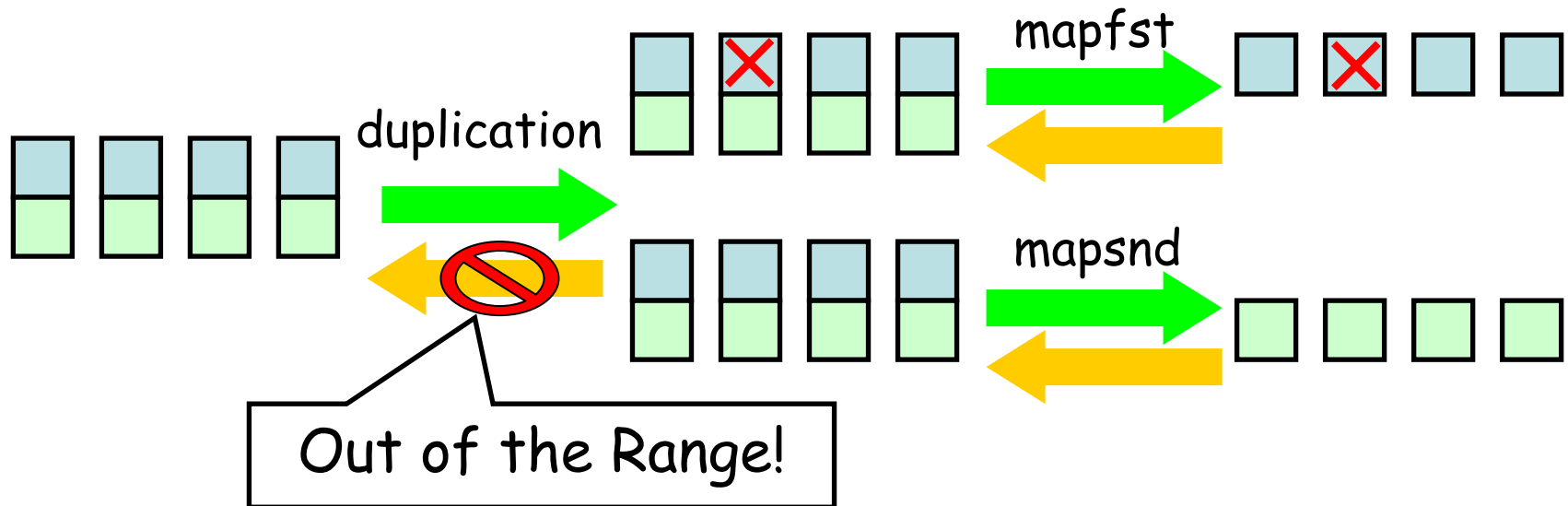
Direct bidirectionalization does not yield useful " f_B " if " f " contains **duplications**.

Review: "unzip" example

- `unzip(s) = (mapfst(s), mapsnd(s))`

fwd.

```
unzip [(1,"a"),(2,"b"),(3,"c")] = ([1,2,3], ["a","b","c"])
```



Multiple Data Traversals

- Multiple data traversals causes problem.
 - $h(x) = (f(x), g(x))$
 - The input "x" is traversed twice.
 - Problem:
 - How to combine " f_B " and " g_B "?
 - [Hu et al. 04, Mu et al. 04] is not SE-free.

How to avoid use of multiple data traversals in the definition of transformation like "h"?

Tupling Helps!

- Problematic **Multiple Data Traversals**
 - "dup" is not problematic, but
"unzip(x) = (mapfst(x), mapsnd(x))" is.

← **Tupling** [Hu et al. 97, Chin 93]

- eliminates Multiple Data Traversals

```
unzip(x) = (mapfst(x), mapsnd(x))
```

```
unzip([]) = ([], [])
```

```
unzip((a,b):x) =
```

```
  let (s,t) = unzip(x) in (a:s,b:t)
```


Language to be Bidirectionalized

- For functions *AFTER* tupling
- *Treeless* [Wadler 90] except projection


```
unzip([]) = ([], [])  
unzip((a,b):x) = let (s,t) = unzip(x) in (a:s,b:t)
```

```
cb(Z) = (L,L)  
cb(S(x)) = let (s,t) = cb(x) in (N(s,t), N(s,t))
```


○ Output Dup.

 `f(...)` =
let `y` = `f(x)`
z = `f(x)`
in ...

MDT

 `f(...)` =
let `y` = `f(x,x)`
z = `f(w)`
in ...

MDT

 `f(...)` =
let `y` = `f(x)`
z = `f(y)`
in ...

Non-Treeless

Formal Description

- 1st order treeless functional programming language
 - with "let" binds

$$\begin{aligned} \text{programs} & ::= \text{rule}_1 \cdots \text{rule}_n \\ \text{rule} & ::= f(p_1, \dots, p_n) = \text{let binds in } (q_1, \dots, q_m) \\ p, q & ::= C(p_1, \dots, p_n) \mid x \\ \text{binds} & ::= \text{bind}_1 \cdots \text{bind}_n \\ \text{bind} & ::= (y_1, \dots, y_m) = f(x_1, \dots, x_n) \end{aligned}$$

Involved Example

- Extract all the sections containing section titles & add title-list.

```
toc(Doc(x)) = let (y,z) = f(x) in (y,Doc(z))
f([])       = ([],[])
f(SecT(t,b):x) = let (y,z) = f(x) in (t:y,Sec(t,b):z)
f(Sec(b):x)   = let (y,z) = f(x) in (y,z)
```

```
Doc(
  Sec( "Recently, ..." )
  SecT( "Prelim.", "This ..." )
  SecT( "Lang.", "...") )
```

```
([ "Prelim.", "Lang." ],
 Doc(
  SecT( "Prelim.", "This ..." )
  SecT( "Lang.", "...") ))
```

Bidirectionalization

- Constant Complement [Bancilhon&Spyratos 81]
 - "g" is a **complement** func of "f"
 $\Leftrightarrow \langle f, g \rangle$ is injective ($\langle f, g \rangle(x) = (f(x), g(x))$)
 - Complement define U such that
 $S \sim V \times U$ & f_B keeps "U"-part equal.

$$f_B(s, v) = \langle f, f^c \rangle^{-1}(v, f^c(s))$$

Derivation of Complement

- Use **Recursion Structure** of functions.
 - Use syntax instead of semantics.

- Unused Variables

$$\boxed{\text{fst}(x, y) = x} \longrightarrow \boxed{\text{fst}^c(x, y) = y}$$

$$\text{fst}_B((x, y), v) \sim (v, y)$$

- Used Rules (= Recursion Structure)

$$\boxed{\begin{array}{l} \text{add}(Z, y) = y \\ \text{add}(S(x), y) = S(\text{add}(x, y)) \end{array}} \longrightarrow \boxed{\begin{array}{l} \text{add}^c(Z, y) = B_1 \\ \text{add}^c(S(x), y) = B_2(\text{add}^c(x, y)) \end{array}}$$

$$\text{add}_B((x, y), v) \sim (x, v-x) \text{ if } v \geq x$$

Involved Example

$$\begin{aligned} \text{toc}(\text{Doc}(x)) &= \text{let } (y,z) = f(x) \text{ in } (y, \text{Doc}(z)) \\ f([]) &= ([], []) \\ f(\text{SecT}(t,b):x) &= \text{let } (y,z) = f(x) \text{ in } (t:y, \text{Sec}(t,b):z) \\ f(\text{Sec}(b):x) &= \text{let } (y,z) = f(x) \text{ in } (y,z) \end{aligned}$$

$$\begin{aligned} \text{toc}^c(\text{Doc}(x)) &= \text{let } (y,z) = f^c(x) \text{ in } (y,z) \\ f^c([]) &= (B_{11}, B_{11}) \\ f^c(\text{SecT}(t,b):x) &= \text{let } (y,z) = f^c(x) \text{ in } (B_{21}(y), B_{22}(z)) \\ f^c(\text{Sec}(b):x) &= \text{let } (y,z) = f^c(x) \text{ in } (B_{31}(b,y), B_{32}(b,z)) \end{aligned}$$

Doc(
 Sec("Recently, ...")
 SecT("Prelim.", "This ...")
 SecT("Lang.", "..."))



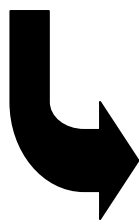
B₃₁ ("Lang.", "...")
 B₃₂ ("Recently, ...")
 B₂₁ ("Prelim.", "This ...")
 B₂₂
 B₂₂
 B₂₂
 B₁₁

Syntax-Based Merits

Loop Optimization Helps

- Linear-recursion optimization by range analysis.

```
zip([], y) = []  
zip(a:x, []) = []  
zip(a:x, b:y) = (a, b) : zip(x, y)
```



```
zipc([], y) = B1(y)  
zipc(a:x, []) = B2(a, x)  
zipc(a:x, b:y) = zipc(x, y)
```

```
zipB((x, y), v) ~ unzip(v) # leftover x y  
where (s, t) # (a, b) = (s++a, t++b)
```

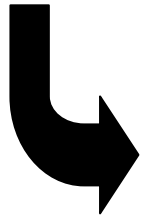
```
leftover [1, 2, 3, 4, 5] [6, 8, 9] = ([4, 5], [])
```

Syntax-Based Merits

Injective Analysis Helps

- Injectivity check by range inference
 - Sound but not complete

```
unzip([]) = ([], [])  
unzip((a,b):x)  
  = let (s,t) = unzip(x) in (a:s,b:t)
```



```
unzipc([]) = (B,B)  
unzipc((a,b):x) = (B,B)
```

```
unzipB(s, (x,y)) ~ zip x y if length x = length y
```

Summary

Impl: <<http://www.ipl.t.u-tokyo.ac.jp/~kztk/b18n2/>>

- Bidirectionalization of Programs with Duplication
 - Two sort of duplications
 - Input (Multiple Data Traversals)/Output
 - "let" clarifies Input/Output.
- Side-Effect tree bwd. trans.
 - Bwd. Trans. =
SE-free + SEfull + Reconciler?
Enlarged!!

Limitation

- Range-overlap check is complete but not sound
 - Injectivity check is sound but not complete.
 - Exact check is undecidable [Fülöp94]

```
addsub(Z, y)      = (y, M(y))
addsub(S(x), Z)   = (S(x), P(S(y)))
addsub(S(x), S(y)) = let (s, t) = addsub(x, y)
                   in (S(S(s)), t)
```

Is there any almost automatic method that proves the injectivity of "addsub"?

Future Work

- Can I use reversible language for user defined complements?
 - Do we make users to write complements directly?
- Range \rightarrow SEfull bwd.&Reconciler?
 - Accumulation-free context-free grammar + **sync**. (cf. Generalized Multitext Grammar [Melamed et al. 04])
e.g.: "unzip"
 - $T \rightarrow ([], [])$
 - $T \rightarrow (\text{Any} : \pi_1 T^{(1)}, \text{Any} : \pi_2 T^{(1)})$
 - Parsing? It restores Recursion Structure.