In the last bx seminar in Dagstuhl …

Zhenjiang

In get-based bx, there is inherited ambiguity:

many puts may correspond one get

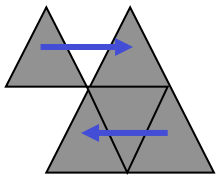
Unless we have a way to choose among these puts, we would come up with an unpredicable bx …

Benjamin

We should be able to remove ambiguity by writing put!

Zhenjiang

Trying to code some bx combinators for writing put in Curry, and discussed it a bit with Soichiro, Jeremy, Janis …



Validation of BX Programs

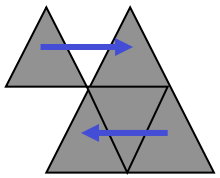
Well-behavedness of Treeless Putback Definitions for Bidirectional Programming is Decidable

Zhenjiang Hu (NII)

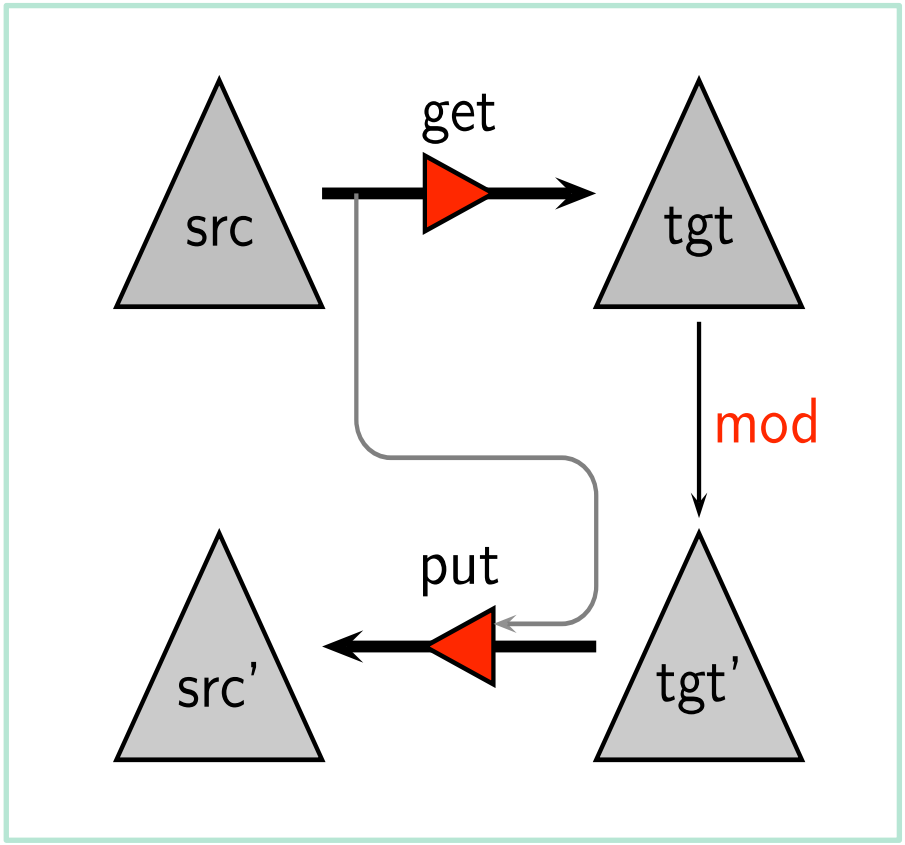
Joint work with

Hugo Pacheco and Sebastian Fischer

December 2013



Bidirectional Transformation

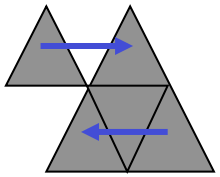


Get-Put:

put s (get s) = s

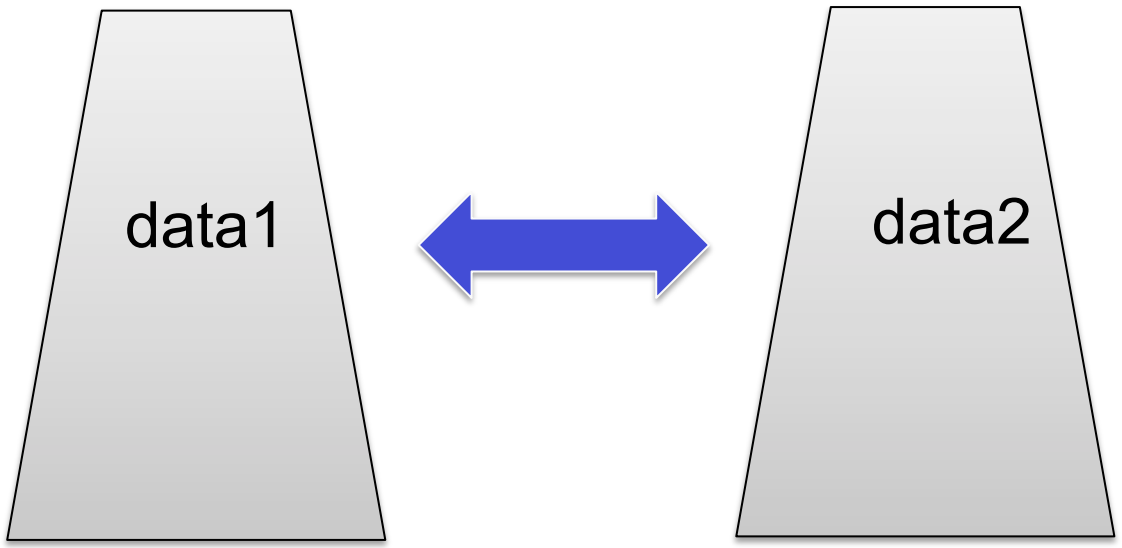
Put-Get:

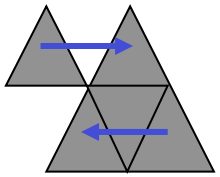
get (put s t) = t



What is BX Programming?

Define a pair of functions get/put to synchronize two kinds of data.

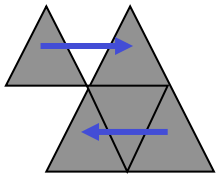




What is BX Programming?

Define a pair of functions get/put to synchronize two kinds of data.

Define a well-behaved put to synchronize two kinds of data.



Well-behaved "put"

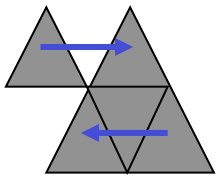
Definition: A "put" function is said to be well-behaved, if there exists a (unique) "get" function such that GetPut and PutGet hold.

Question: Are the following put functions well behaved?

- put1 s v = s
- put2 s v = 1 : v
- put3 [] v = v

  put3 (a : s) v = a : v

Difficult to check because we do not have "get" yet ...



Well-behaved "put"

Lemma:

Put is well-behaved, iff

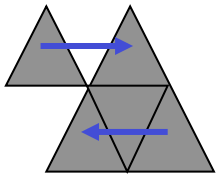
1. View-deterministic

   put s1 v1 = put s2 v2 → v1 = v2

2. View-stable

   for any s, there exists a v, such that put s v = s

Reference:
Sebastian Fischer, Zhenjiang Hu, Hugo Pacheco,
A Clear Picture of Lenses,
(to be submitted, available upon request)

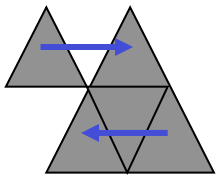


Languages for Putback Programming

A treeless language for define primitive well-behaved puts.

+

A set of combinators to compose smaller well-behaved puts to form bigger ones



A Treeless Language *PDL*

A Treeless Language for Put-based Bidirectional Programming

Rule

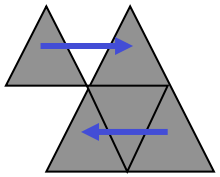
$$f\ p_s\ p_v = t$$

Treeless Term

$$\begin{array}{lll} t ::= v & & \{ \text{ variable } \} \\ \quad | \ \ C\ t_1 \ \ldots \ t_n & & \{ \text{ constructor application } \} \\ \quad | \ \ f\ x_s\ x_v & & \{ \text{ put application } \} \end{array}$$

Pattern

$$\begin{array}{lll} p ::= \ x & & \{ \text{ variable } \} \\ \quad | \ \ x\ @\ p & & \{ \text{ look-ahead variable } \} \\ \quad | \ \ C\ p_1 \ \ldots \ p_n & & \{ \text{ constructor pattern } \} \end{array}$$



Syntactic Assumptions

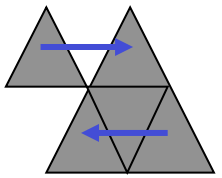
- Affine: each variable appears at most once in rhs

```
put (s:ss) vs = s : vs              GOOD
put (s:ss) vs = s : (vs++vs)        BAD
```

- Structured: recursive calls are on smaller sub-patterns

```
put (s:ss) (v:vs) = v : put ss vs   GOOD
put ss (v:vs) = v : put ss vs       GOOD
put ss vs = 1 : put ss vs           BAD
put (s:ss) (v:vs) = v : put vs ss   BAD
```

- Total: patterns are exhausted

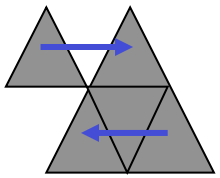


Example

putAs [A 1, A 2, B 3, A 4] [10, 11,12] ➔ [A 10, A 11, B 3, A 12]
putAs [A 1, A 2, B 3, A 4] [10, 11] ➔ [A 10, A 11, B 3]
putAs [A 1, A 2, B 3, A 4] [10, 11,12,13] ➔ [A 10, A 11, B 3, A 12, A 13]

```
putAs []          []          = []
putAs (ss@[])     (v:vs)      = A v : putAs ss vs
putAs (A a : ss)  (vs@[])     = putAs ss vs
putAs (A a : ss)  (v : vs)    = A v : putAs ss vs
putAs (B b : ss)  vs          = B b : putAs ss vs
```

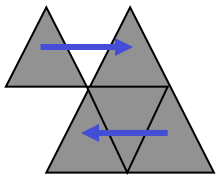
Affine, structured, total



Example

putAs [A 1, A 2, B 3, A 4] [10, 11,12] ➔ [A 10, A 11, B 3, A 12]
putAs [A 1, A 2, B 3, A 4] [10, 11] ➔ [A 10, A 11, B 3, B 4]
putAs [A 1, A 2, B 3, A 4] [10, 11,12,13] ➔ [A 10, A 11, B 3, A 12, B 0, A 13]

```
putAs []          []          = []
putAs (ss@[])     (v:vs)      = A v : B 0 : putAs ss vs
putAs (A a : ss)  (vs@[])     = B a : putAs ss vs
putAs (A a : ss)  (v : vs)    = A v : putAs ss vs
putAs (B b : ss)  vs          = B b : putAs ss vs
```

Main Results

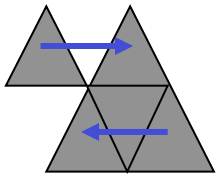
Theorem:

Well-behavedness of a put defined in $\mathcal{PDL}$ is decidable.

Validation Algorithm:

(Soundness): A valid put is well-behaved.

(Completeness): Any well-behaved put is valid.



View-Determination Validation

Lemma:

Put is well-behaved, iff

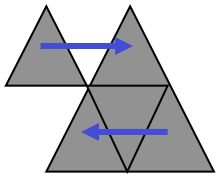
1. View-deterministic

   put s1 v1 = put s2 v2 ➔ v1 = v2

2. View-stable

   for any s, there exists a v, such that put s v = s

The relation from updated sources to views forms a total function.

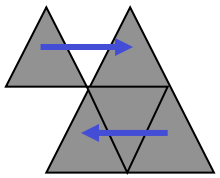


View-Determination Validation

The relation from updated sources to views forms a total function.

(1) The relation R can be automatically derived from the put defined in $\mathcal{PDL}$, which is a finite tree transducer.

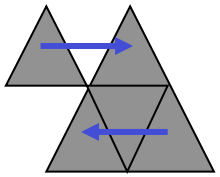
(2) FACT: Single-valuedness of finite tree transducers is decidable (Seidl:TCS92)



View-Stability Validation

Lemma:
Put is well-behaved, iff
1. View-deterministic
   put s1 v1 = put s2 vs ➔ v1 = v2
2. View-stable
   for any s, there exists a v, such that put s v = s

[v can only be R(s) from view-determination]
Let h x y = put x (R y). The validation of h s s = s is decidable.



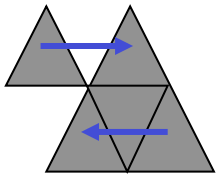
View-Stability Validation

[v can only be R(s) from View-determination]

Let h x y = put x (R y). The validation of h s s = s is decidable.

(1) h is of treeless form in $\mathcal{PDL}$. (h is a provable convergent complete constructor rewriting system (CS))

(2) For a CS, the inductive validity of h t1 t2 = p is decidable (so does h s s = s) [Giesl&Kapur: IJCAR01]



Conclusion

Main Result for BX Program Validation:

Well-behavedness of Treeless Putback Definitions for Bidirectional Programming is Decidable

Todo:

provide a practical put-based programming language

- Easy to code
- Easy to debug
- Easy to Optimize

New post-docs are welcome!