Functional Logic Semantic Bidirectionalization for Free!

Hugo Pacheco

HASLab
INESC TEC & Universidade do Minho, Braga, Portugal

FATBIT/SSaaPP Workshop
Braga - September 18th 2012

Towards Functional Logic Semantic Bidirectionalization for Free!

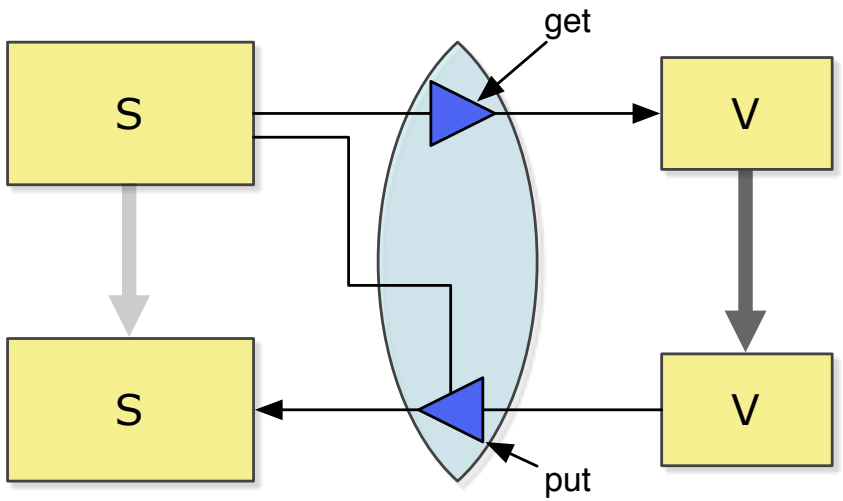
Hugo Pacheco

HASLab
INESC TEC & Universidade do Minho, Braga, Portugal

FATBIT/SSaaPP Workshop
Braga - September 18th 2012

BXs and Lenses

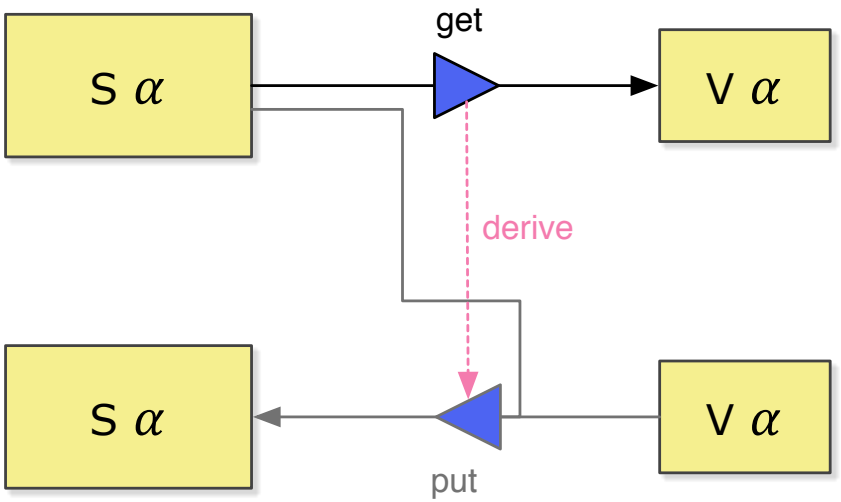
- lenses are one of the most popular BX frameworks



- existing lens systems vary on the bidirectionalization approach
- how is a lens derived from a specification?

Functional Semantic Bidirectionalization

- Voigtländer proposed the semantic bidirectionalization of Haskell functions [POPL'09]

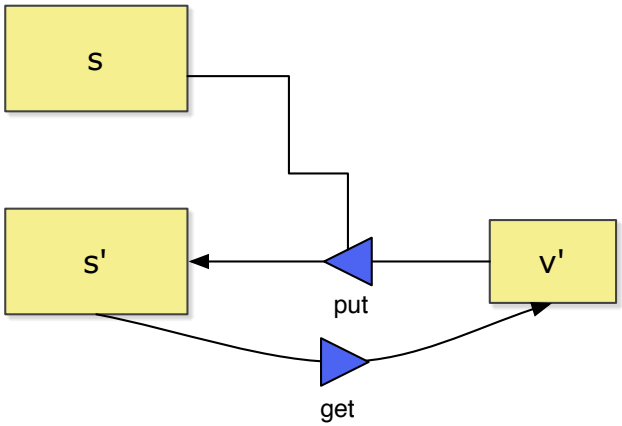


- *put* via the polymorphic interpretatation of *get*
- limited expressiveness - only polymorphic *get* functions
- limited updatability - even for the mixed approach of Voigtländer et al. [ICFP'10]

The Lens Laws

- PutGet law

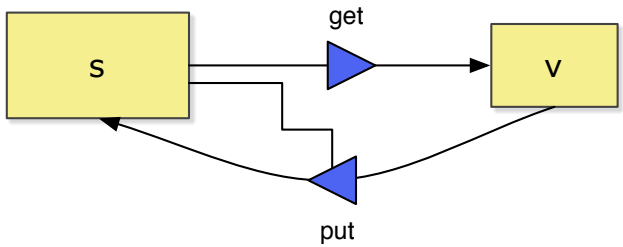
  put must translate view updates exactly.



$$get\ (put\ v'\ s) \sqsubseteq v'$$

- GetPut law

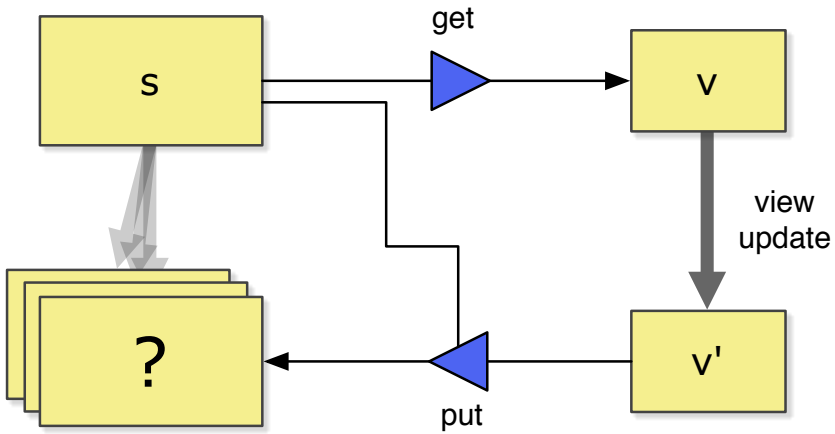
  put must preserve empty view updates.



$$put\ (get\ s)\ s \sqsubseteq s$$

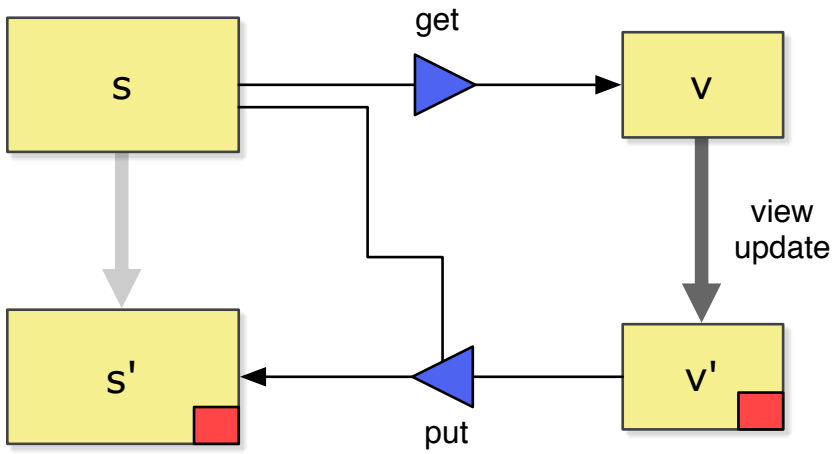
A Better GetPut Law?

- when the view is modified there are many source updates
- anything can happen! - no restriction on the permitted translations



A Better GetPut Law?

- when the view is modified there are many source updates
- only "good" can happen - only minimal source updates are permitted



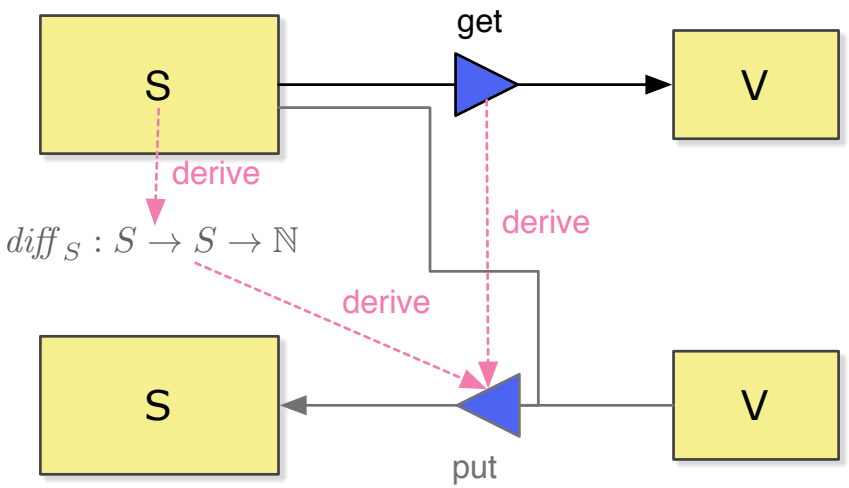
$$\forall\ v', s, s'.\ diff\ (put\ v'\ s)\ s\ \leqslant\ diff\ s'\ s \qquad \text{PutDiff}$$

- the differencing function depends on the source type

$$diff_S : S \to S \to \mathbb{N}$$

Functional Logic Semantic Bidirectionalization

- Idea: use Curry, a functional logic programming language, to compute such minimal updates
- functional programming: Haskell-like syntax
- logic programming: logic variables, built-in search (findall, best)



- derive *diff* from the source type
- derive *put* from *get* and *diff*

A *diff* for Algebraic Data Types

- a generic *diff* for algebraic data types

  Eelco Lempsink, Sean Leather and Andres Löh
  Type-Safe Diff for Families of Datatypes
  Workshop on Generic Programming 2009.

- we implement this *diff* in Curry

$$
\begin{aligned}
&\mathit{diffND}_{List} : [a] \to [a] \to \mathbb{N} \\
&\mathit{diffND}_{List}\ [\,]\ [\,] = 0 \\
&\mathit{diffND}_{List}\ [\,]\ (y : ys) = 1 + \mathit{diffND}_{List}\ [\,]\ ys \quad \text{-- insert} \\
&\mathit{diffND}_{List}\ (x : xs)\ [\,] = 1 + \mathit{diffND}_{List}\ xs\ [\,] \quad \text{-- delete} \\
&\mathit{diffND}_{List}\ (x : xs)\ (y : ys) \\
&\quad \mid x =:= y \quad = \mathit{diffND}_{List}\ xs\ ys \quad \text{-- copy} \\
&\quad \mid x =/= y = 1 + \mathit{diffND}_{List}\ (x : xs)\ ys \quad \text{-- insert} \\
&\qquad\qquad\quad ?\ 1 + \mathit{diffND}_{List}\ xs\ (y : ys) \quad \text{-- delete}
\end{aligned}
$$

$$
\mathit{diff}_{List}\ s'\ s = \mathit{unpack}\ \$\ \mathit{head}\ \$\ \mathit{best}\ (\lambda n \to \mathit{diffND}\ s'\ s =:= n)\ (\leqslant)
$$

- this *diff* calculates the sequence of insert, delete and copy operations with the minimal cost

Curry Implementation

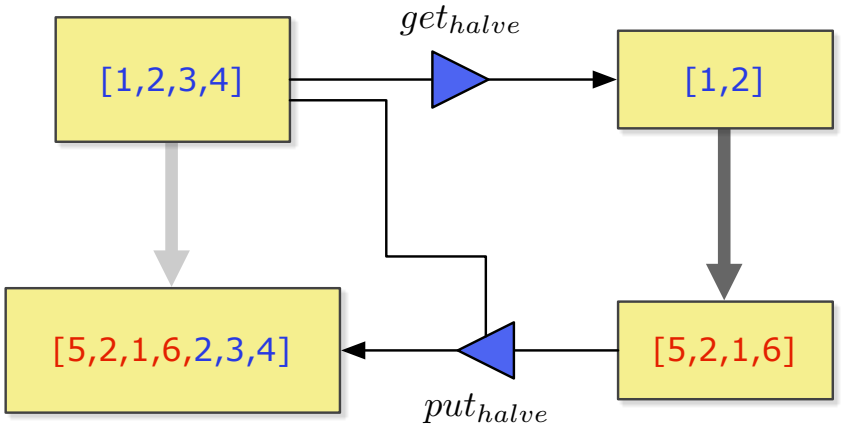
- we implement *put* in Curry as a non-deterministic function

$$
\begin{aligned}
&put :: V \to S \to S \\
&put\ v'\ s = putn\ n\ v'\ s =:= s' \\
&\quad \textbf{where}\ n = diff'_S\ v'\ s \\
&\qquad\quad s'\ free \\
&diff'_S\ v'\ s = unpack\ \$\ head\ \$\ best \\
&\quad (\lambda n \to \textbf{let}\ s'\ free\ \textbf{in}\ get\ s' =:= v'\ \&\ diffND_S\ s'\ s =:= n) \\
&putn :: \mathbb{N} \to V \to S \to S \\
&putn\ n\ v'\ s \mid get\ s' =:= v'\ \&\ diff_S\ s'\ s =:= n = s'\ \textbf{where}\ s'\ free
\end{aligned}
$$

1. calculate the minimal difference between any new source (whose view is v') and the original source s
2. return any new source whose difference to the original source s is n

Example 1 (*halve*) - First Attempt

- calculate the first half of a list



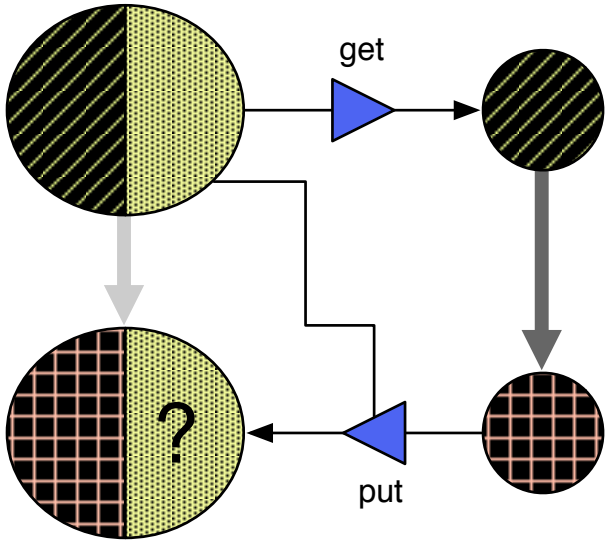
$$get = halve$$

$$\begin{aligned}
&halve :: [a] \rightarrow [a] \\
&halve\ [\,] = [\,] \\
&halve\ (x : xs) = x : halve'\ xs\ xs \\
&\quad \textbf{where}\ halve'\ xs\ [\,] = [\,] \\
&\qquad halve'\ xs\ [y] = [\,] \\
&\qquad halve'\ (x : xs)\ (y : z : zs) = x : halve'\ xs\ zs
\end{aligned}$$

- is this the best result?

Calculating a View Complement

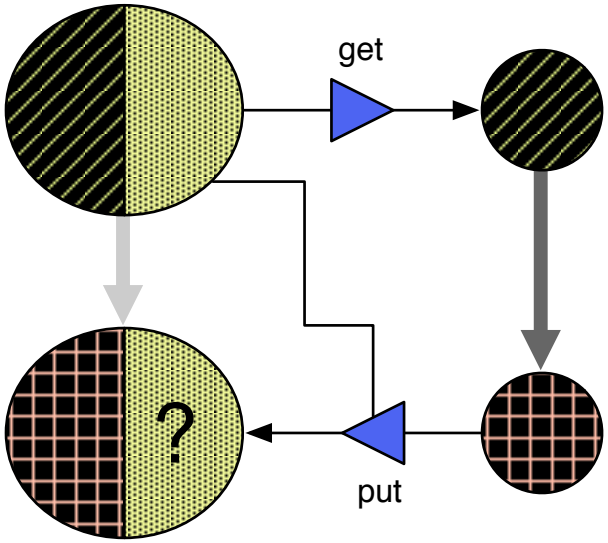
- view complement = source data not present in the view

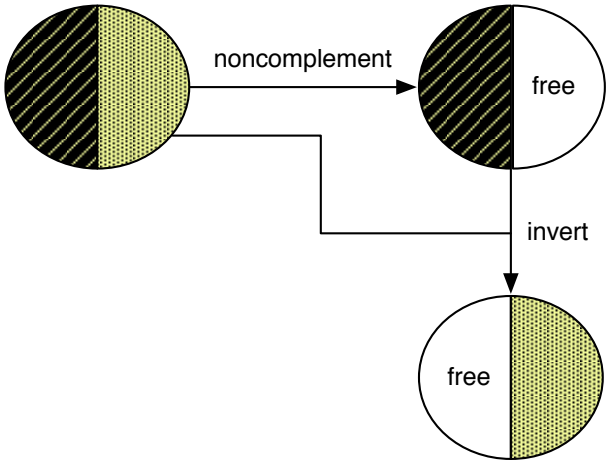


- *put* should only recover data from the view complement

Calculating a View Complement

- view complement = source data not present in the view





- *put* should only recover data from the view complement
- how to calculate the complement in Curry?

$$noncomplement_S\ s = findfirst\ (\lambda x \rightarrow get\ x =:= get\ s\ \&\ match_S\ x\ s)$$

$$complement_S\ s = invert_S\ s\ (noncomplement_S\ s)$$

Calculating a View Complement

- we refine *diff* to take into account the source complement

$$
\begin{aligned}
&\mathit{diffND}_{List} : [a] \to [(a,a)] \to \mathbb{N} \\
&\mathit{diffND}_{List}\ [\,]\ [\,] = 0 \\
&\mathit{diffND}_{List}\ [\,]\ ((v,y) : ys) = \mathit{insert} \\
&\mathit{diffND}_{List}\ (x : xs)\ [\,] = \mathit{delete} \\
&\mathit{diffND}_{List}\ (x : xs)\ ((v,y) : ys) \\
&\quad \mid x =:= y\ \ \&\ \mathit{isVar}\ v =:= \mathit{False} = \mathit{copy} \\
&\quad \mid x =:= y\ \&\ \mathit{isVar}\ v =:= \mathit{True}\ \ = \mathit{insert}\ ?\ \mathit{delete} \\
&\quad \mid x = /= y \qquad\qquad\qquad\quad = \mathit{insert}\ ?\ \mathit{delete} \\
&\mathit{diff}_{List}\ s'\ s = \mathit{unpack}\ \$\ \mathit{head}\ \$\ \mathit{best}\ (\lambda n \to \mathit{diffND}\ s'\ \mathit{cs} =:= n)\ (\leqslant) \\
&\quad \textbf{where}\ \ \mathit{cs} = \mathit{zip}\ (\mathit{complement}\ s)\ s
\end{aligned}
$$

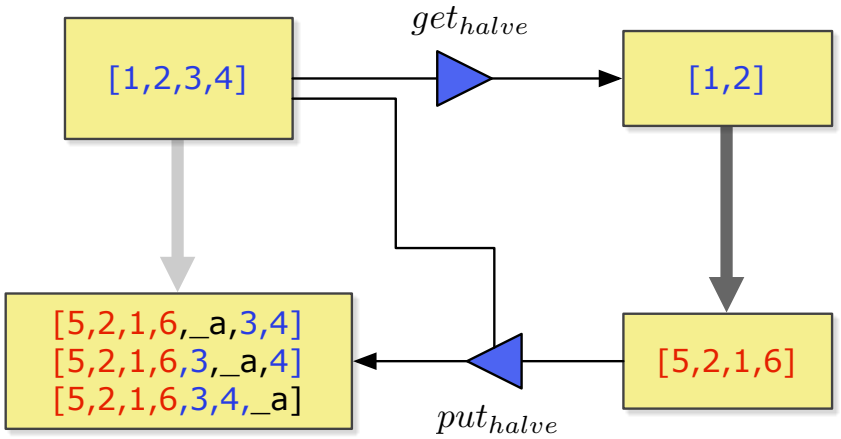
- we do not allow source data not in the complement to be copied

Example 1 (*halve*) - Second Attempt

- calculate the complement of the source

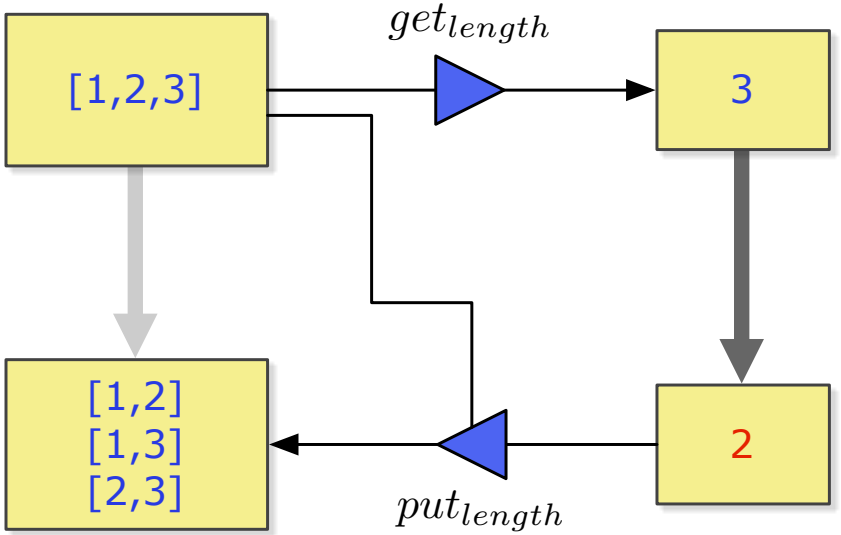
$$noncomplement_{List}\ [1,2,3,4] = [1,2,\_a,\_b]$$
$$complement_{List}\ [1,2,3,4] = [\_a,\_b,3,4]$$

- calculate the first half of a list (revisited)



Example 2 (*length*)

- compute the length of a list

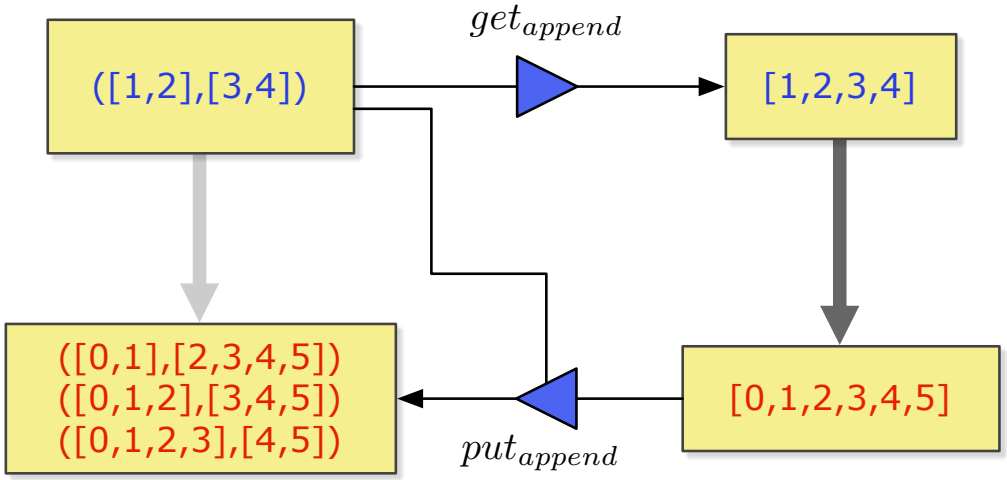


$$get = length$$

$$length :: [a] \rightarrow \mathbb{N}$$
$$length\ [\,] = 0$$
$$length\ (x : xs) = 1 + length\ xs$$

Example 3 (*append*)

- append two lists into a single list



$$get = append$$

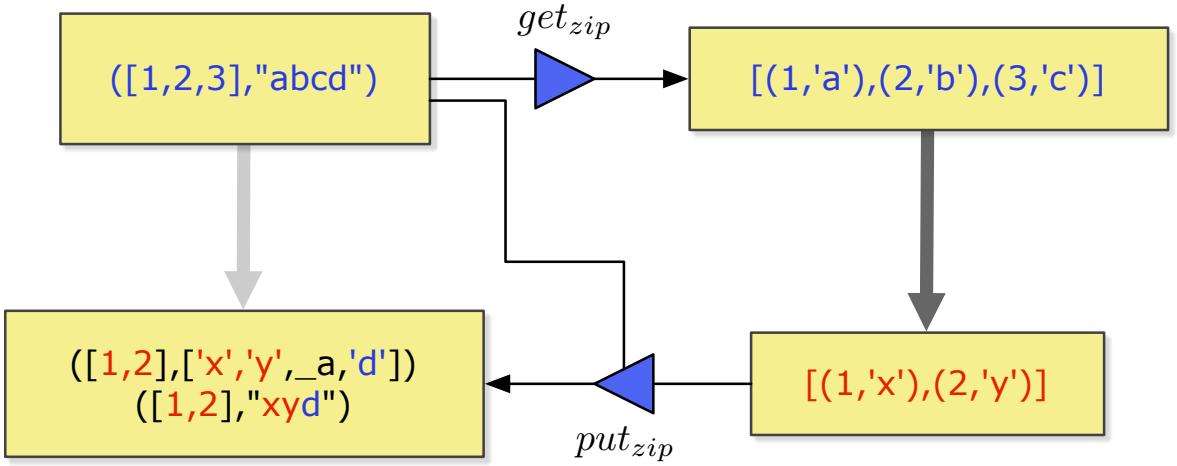
$$append :: ([a], [a]) \rightarrow [a]$$
$$append\ ([\,], ys) = ys$$
$$append\ (x : xs, ys) = x : append\ (xs, ys)$$

- *diff* for pairs of lists

$$diff_{\,List \times List} :: ([a], [a]) \rightarrow ([a], [a]) \rightarrow \mathbb{N}$$

Example 4 (*zip*)

- join the elements of two lists pair-wise into a single list

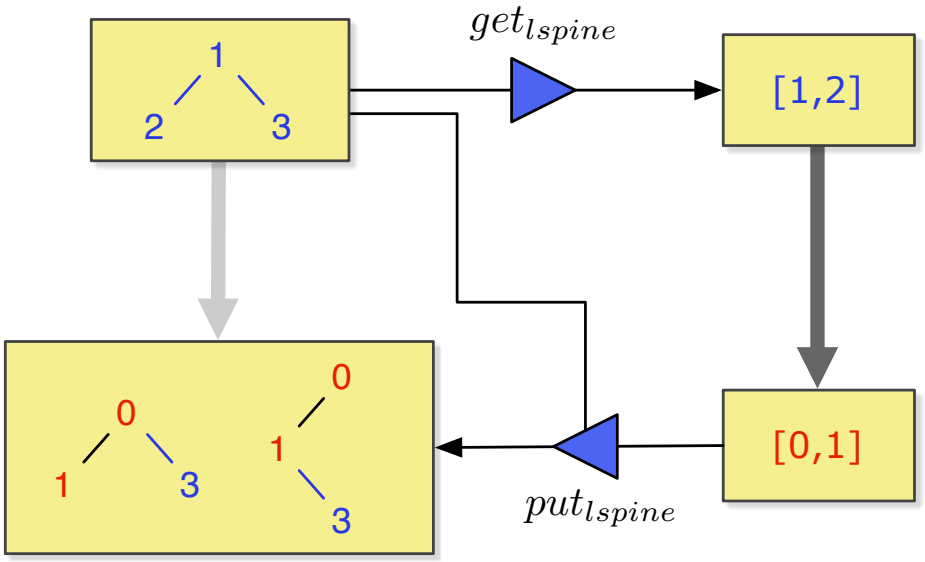


$$get = zip$$

$$zip :: ([a], [b]) \rightarrow [(a, b)]$$
$$zip\ ([\,], ys) = [\,]$$
$$zip\ (xs, [\,]) = [\,]$$
$$zip\ (x : xs, y : ys) = (x, y) : zip\ (xs, ys)$$

Example 5 (*lspine*)

- calculate the left spine of a binary tree



data *Tree a* = *Empty* | *Node a* (*Tree a*) (*Tree a*)

get = *lspine*

lspine :: *Tree a* → [*a*]
lspine Empty = [ ]
lspine (*Node x l r*) = *x* : *lspine l*

Conclusions

- a semantic bidirectionalization approach using Curry
- users define any $get : S \to V$ function, and we derive:
  - a $diff_S$ function
  - a non-deterministic $put : V \to S \to S$ function that lazily returns the "best" new sources

Scoreboard:

+ expressivness
+ updatability
+ properties
– efficiency

Future Work:

- automate the tool
- improve the reduction strategy for infinite search spaces (e.g. *filter*)
- improve *diff*