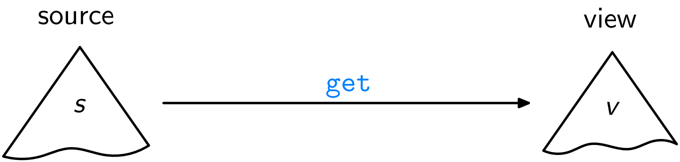
Semantic Bidirectionalisation

Janis Voigtländer

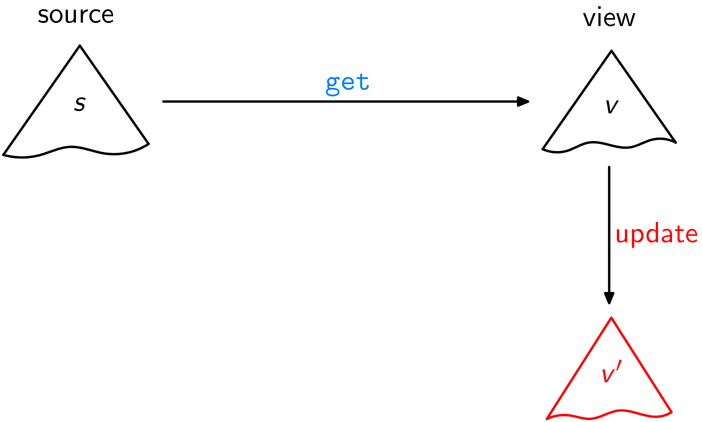
University of Bonn

October 18th, 2010

Bidirectional Transformation



Bidirectional Transformation



source

view

s

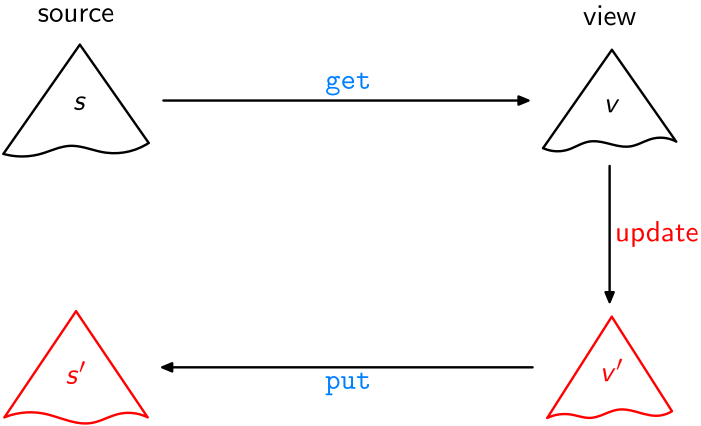
get

v

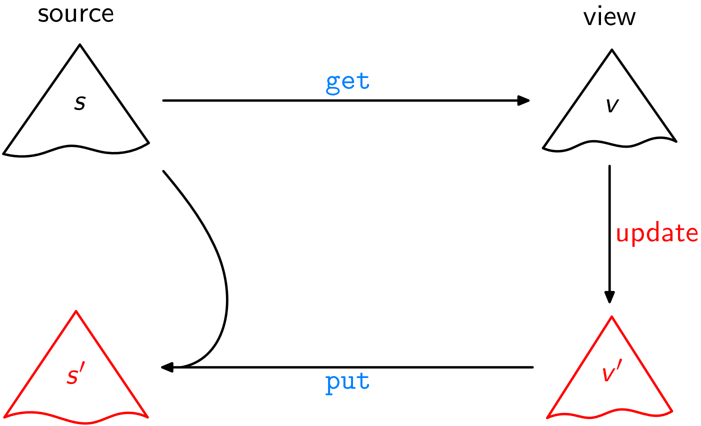
update

v'

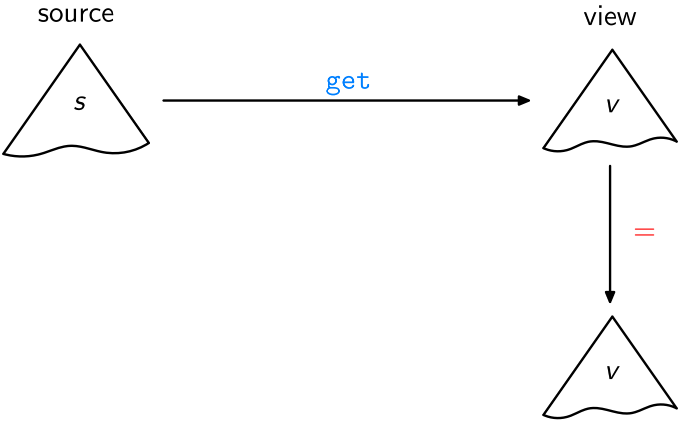
Bidirectional Transformation



Bidirectional Transformation

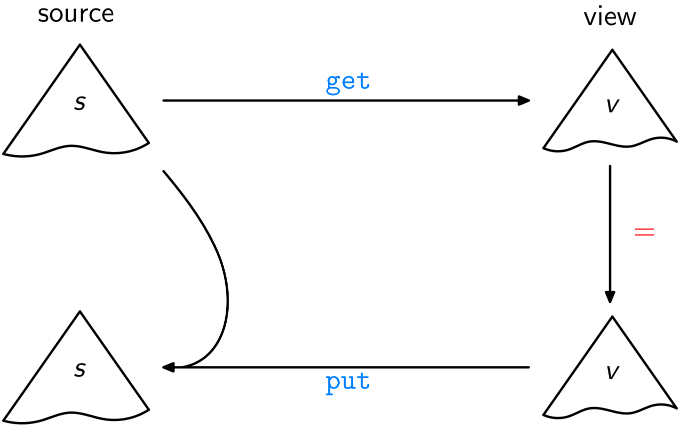


Bidirectional Transformation



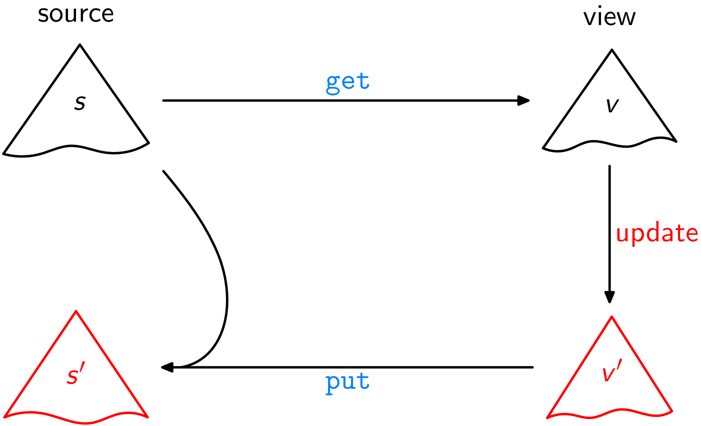
Acceptability / GetPut

Bidirectional Transformation



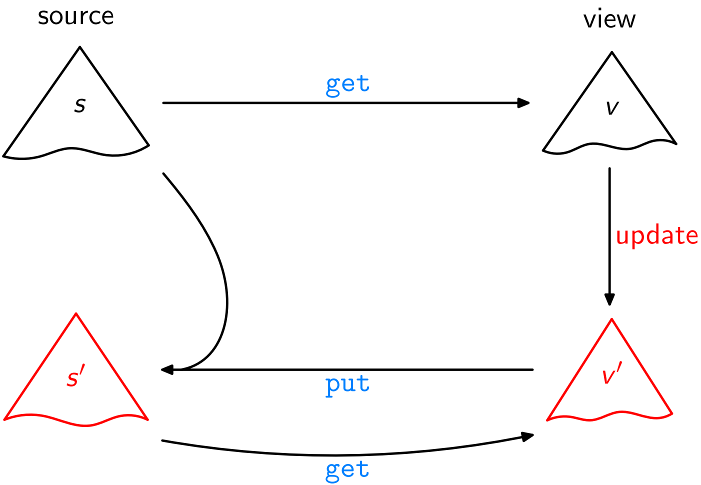
Acceptability / GetPut

Bidirectional Transformation



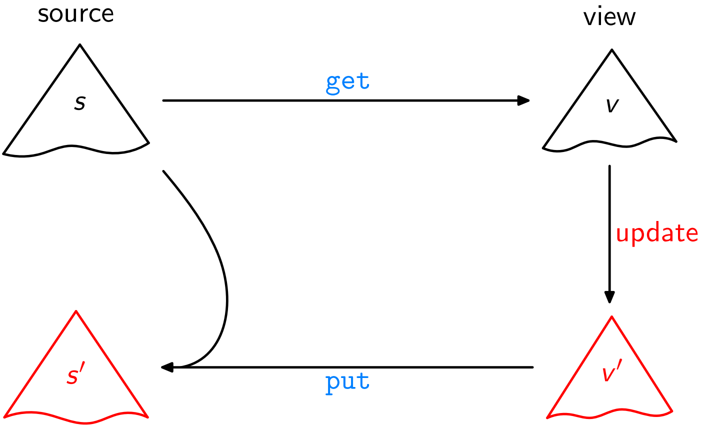
Consistency / PutGet

Bidirectional Transformation



Consistency / PutGet

Bidirectional Transformation



source

view

s

get

v

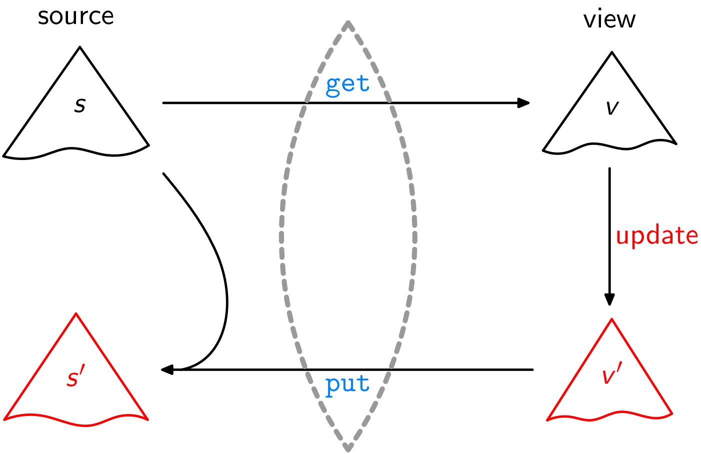
update

s'

put

v'

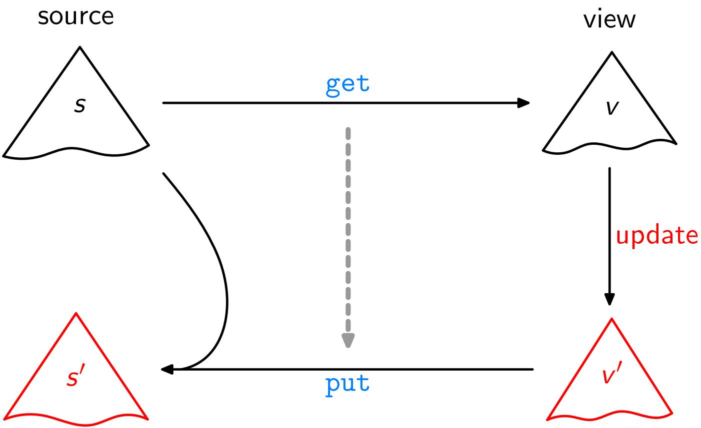
Bidirectional Transformation



Lenses, DSLs

[Foster et al. 2007]

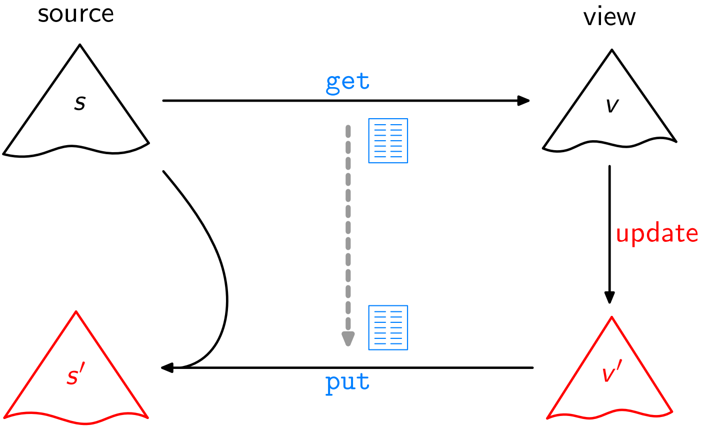
Bidirectional Transformation



Bidirectionalisation

[Matsuda et al. 2007]

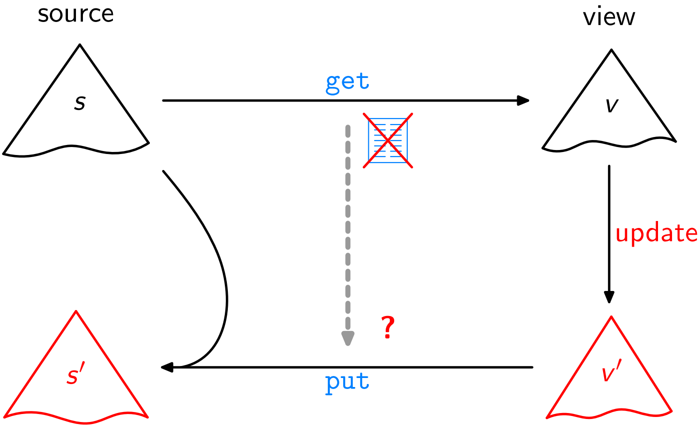
Bidirectional Transformation



Syntactic Bidirectionalisation

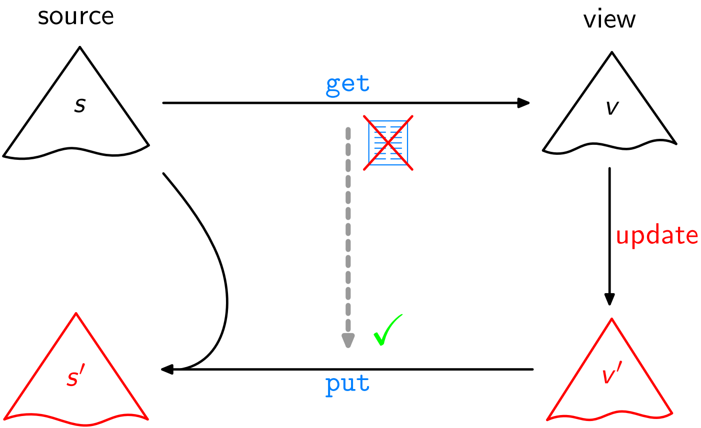
[Matsuda et al. 2007]

Bidirectional Transformation



Semantic Bidirectionalisation

Bidirectional Transformation



Semantic Bidirectionalisation

[V. 2009]

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff` such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

---

[1] "Bidirectionalisation for free!"

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

$$\begin{tikzcd} \text{“abc”} \arrow[r, "\texttt{tail}"] & \text{“bc”} \end{tikzcd}$$

---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

$$\begin{tikzcd}
\text{“abc”} \arrow[r, "\texttt{tail}"] & \text{“bc”} \arrow[d, "\text{update}"] \\
 & \text{“de”}
\end{tikzcd}$$

---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

$$\begin{tikzcd}
\text{“abc”} \arrow[r, "\texttt{tail}"] \arrow[d, bend left] & \text{“bc”} \arrow[d, "\text{update}"] \\
\text{“ade”} & \text{“de”} \arrow[l, "\texttt{bff tail}"]
\end{tikzcd}$$

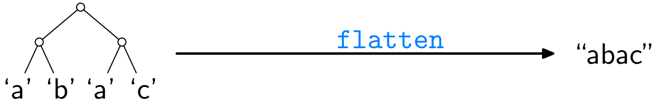
---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:



---

[1] "Bidirectionalisation for free!"

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

‘a’ ‘b’ ‘a’ ‘c’

flatten

“abac”

update

“abxc”

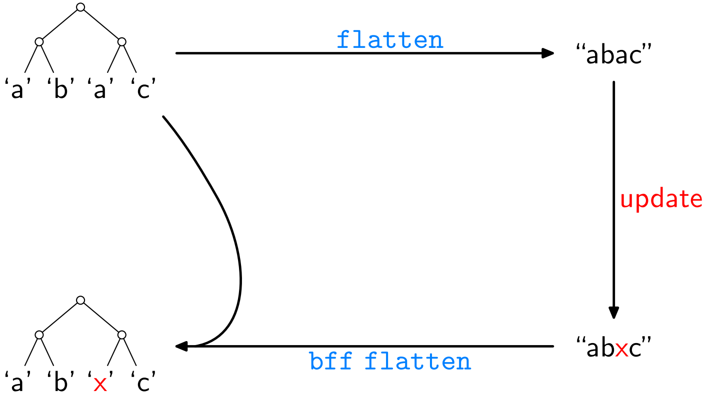
---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:



‘a’ ‘b’ ‘a’ ‘c’

`flatten`

“abac”

update

“abxc”

`bff flatten`

‘a’ ‘b’ ‘x’ ‘c’

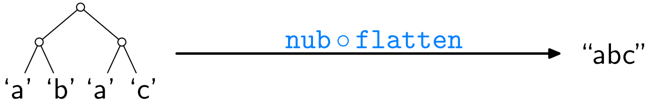
---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:



‘a’ ‘b’ ‘a’ ‘c’ $\xrightarrow{\texttt{nub} \circ \texttt{flatten}}$ “abc”

---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

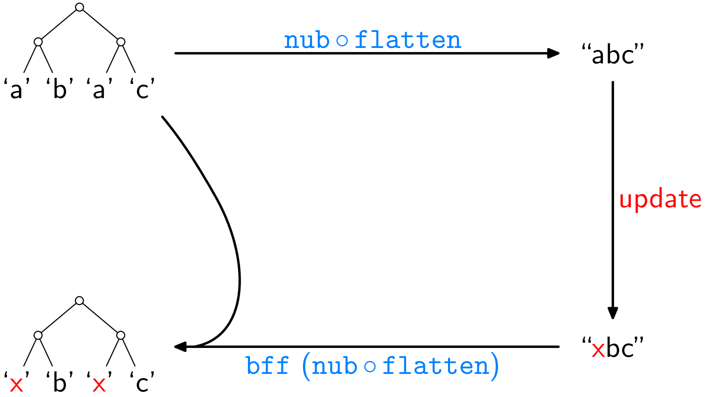
---

[1] “Bidirectionalisation for free!”

Semantic Bidirectionalisation

Aim: Write a higher-order function `bff`[1] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:



---

[1] “Bidirectionalisation for free!”

Analysing Specific Instances

Assume we are given some

$$\texttt{get} :: [\alpha] \to [\alpha]$$

How can we, or `bff`, analyse it without access to its source code?

Analysing Specific Instances

Assume we are given some

$$\texttt{get} :: [\alpha] \to [\alpha]$$

How can we, or `bff`, analyse it without access to its source code?

Idea: How about applying `get` to some input?

Analysing Specific Instances

Assume we are given some

$$\texttt{get} :: [\alpha] \rightarrow [\alpha]$$

How can we, or `bff`, analyse it without access to its source code?

Idea: How about applying `get` to some input?

Like:

$$\texttt{get}\ [0..n] = \begin{cases} [1..n] & \text{if } \texttt{get} = \texttt{tail} \\ [n..0] & \text{if } \texttt{get} = \texttt{reverse} \\ [0..(\texttt{min}\ 4\ n)] & \text{if } \texttt{get} = \texttt{take}\ 5 \\ & \vdots \end{cases}$$

Analysing Specific Instances

Assume we are given some

$$\texttt{get} :: [\alpha] \to [\alpha]$$

How can we, or `bff`, analyse it without access to its source code?

Idea: How about applying `get` to some input?

Like:

$$\texttt{get}\ [0..n] = \begin{cases} [1..n] & \text{if } \texttt{get} = \texttt{tail} \\ [n..0] & \text{if } \texttt{get} = \texttt{reverse} \\ [0..(\texttt{min}\ 4\ n)] & \text{if } \texttt{get} = \texttt{take}\ 5 \\ & \vdots \end{cases}$$

Then transfer the gained insights to source lists other than $[0..n]$!

Using a Free Theorem [Wadler 1989]

For every

$$\mathtt{get} :: [\alpha] \to [\alpha]$$

we have

$$\mathtt{map}\ f\ (\mathtt{get}\ l) \quad = \quad \mathtt{get}\ (\mathtt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{aligned}
&\mathtt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\mathtt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\mathtt{map}\ f\ (a : as) = (f\ a) : (\mathtt{map}\ f\ as)
\end{aligned}$$

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Given an arbitrary list s of length $n+1$, set $l = [0..n]$, $f = (s\,!!)$, leading to:

$$\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n]) \quad = \quad \texttt{get}\ (\texttt{map}\ (s\,!!)\ [0..n])$$

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\quad = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Given an arbitrary list s of length $n+1$, set $l = [0..n]$, $f = (s\,!!)$, leading to:

$$\begin{aligned}
\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n]) \quad &= \quad \texttt{get}\ (\underbrace{\texttt{map}\ (s\,!!)\ [0..n]}_{s}) \\
&= \quad \texttt{get}\ \qquad\qquad s
\end{aligned}$$

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

for arbitrary f and l, where

$$\begin{aligned}
&\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\
&\texttt{map}\ f\ [\,] \qquad\ \ = [\,] \\
&\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as)
\end{aligned}$$

Given an arbitrary list s of length $n+1$,

$$\texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n])$$

$$= \quad \texttt{get}\ s$$

Using a Free Theorem [Wadler 1989]

For every

$$\texttt{get} :: [\alpha] \to [\alpha]$$

we have

$$\texttt{map}\ f\ (\texttt{get}\ l) \quad = \quad \texttt{get}\ (\texttt{map}\ f\ l)$$

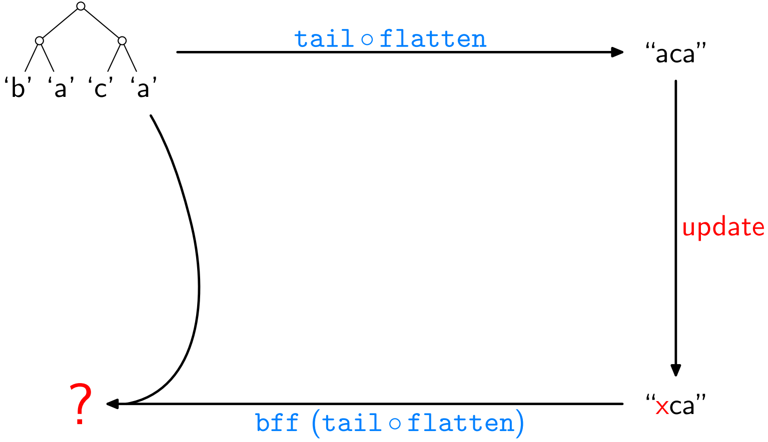
for arbitrary f and l, where

$$\begin{aligned} &\texttt{map} :: (\alpha \to \beta) \to [\alpha] \to [\beta] \\ &\texttt{map}\ f\ [\,] \hphantom{(a : as)} = [\,] \\ &\texttt{map}\ f\ (a : as) = (f\ a) : (\texttt{map}\ f\ as) \end{aligned}$$

Given an arbitrary list s of length $n + 1$,

$$\texttt{get}\ s \quad = \quad \texttt{map}\ (s\,!!)\ (\texttt{get}\ [0..n])$$

The Resulting Bidirectionalisation Scheme by Example



‘b’ ‘a’ ‘c’ ‘a’

`tail ∘ flatten`

“aca”

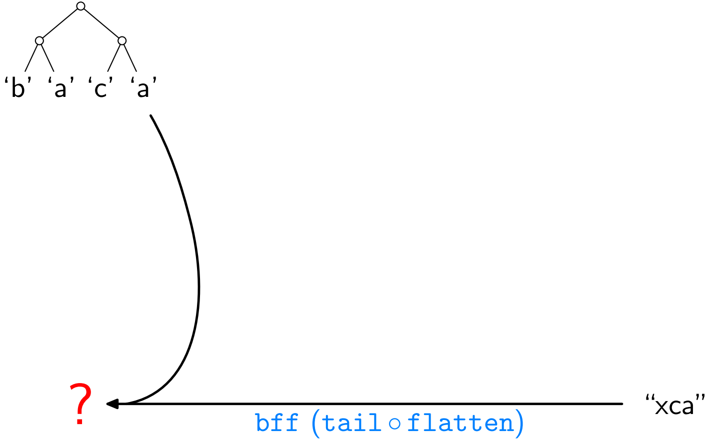
update

“xca”

`bff (tail ∘ flatten)`

?

The Resulting Bidirectionalisation Scheme by Example



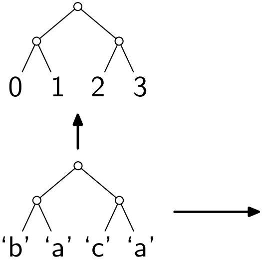
‘b’ ‘a’ ‘c’ ‘a’

?

“xca”

`bff (tail ∘ flatten)`

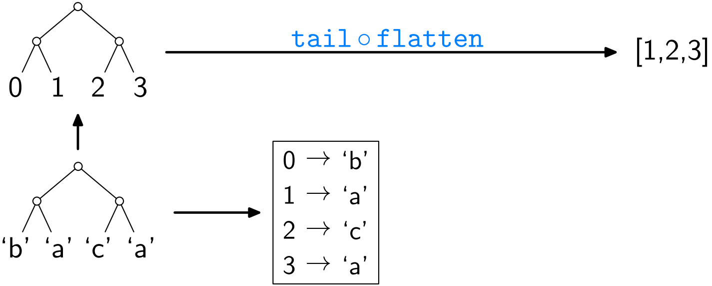
The Resulting Bidirectionalisation Scheme by Example



0 → ‘b’
1 → ‘a’
2 → ‘c’
3 → ‘a’

“xca”

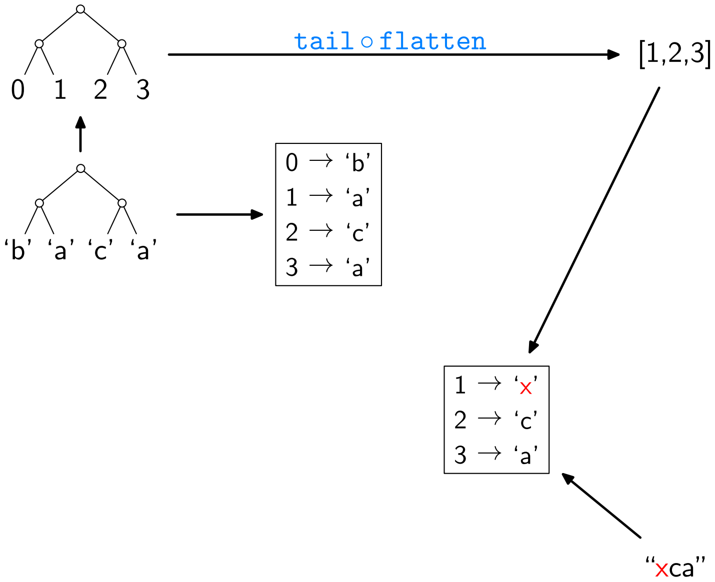
The Resulting Bidirectionalisation Scheme by Example



0 → ‘b’
1 → ‘a’
2 → ‘c’
3 → ‘a’

“xca”

The Resulting Bidirectionalisation Scheme by Example

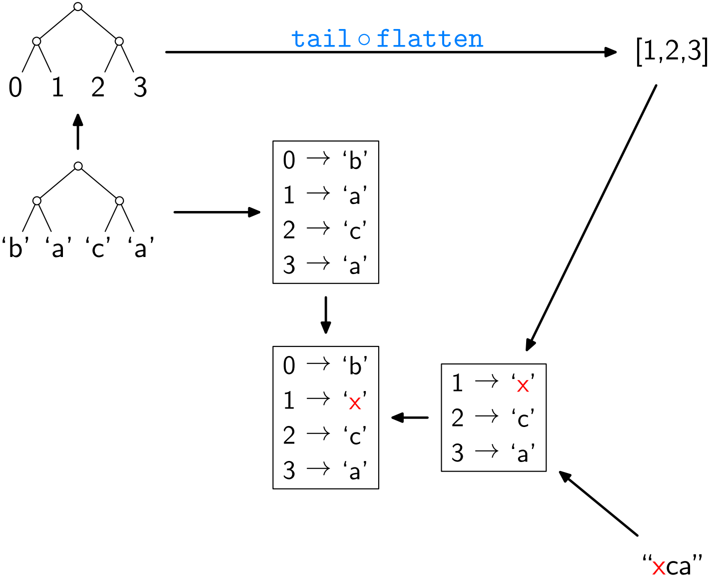


0 → ‘b’
1 → ‘a’
2 → ‘c’
3 → ‘a’

1 → ‘x’
2 → ‘c’
3 → ‘a’

“xca”

The Resulting Bidirectionalisation Scheme by Example



0 1 2 3

`tail ∘ flatten`

[1,2,3]

‘b’ ‘a’ ‘c’ ‘a’

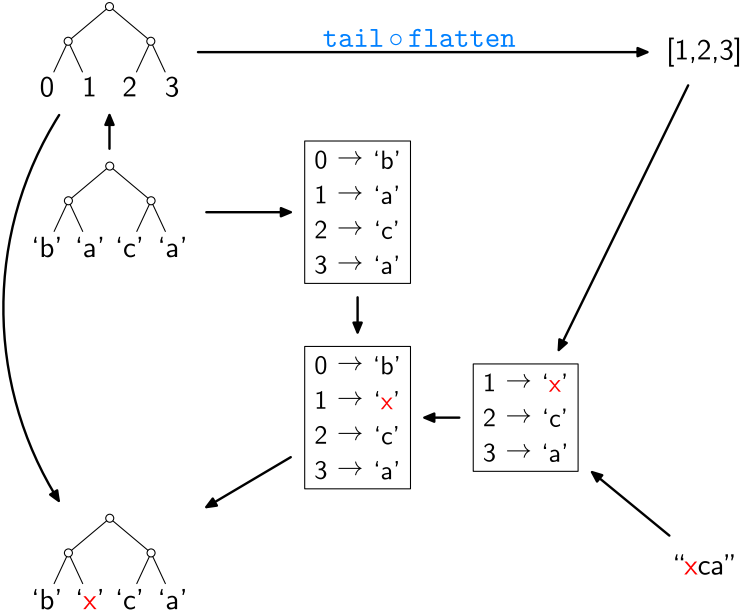
0 → ‘b’
1 → ‘a’
2 → ‘c’
3 → ‘a’

0 → ‘b’
1 → ‘x’
2 → ‘c’
3 → ‘a’

1 → ‘x’
2 → ‘c’
3 → ‘a’

“xca”

The Resulting Bidirectionalisation Scheme by Example



`tail ∘ flatten`

[1,2,3]

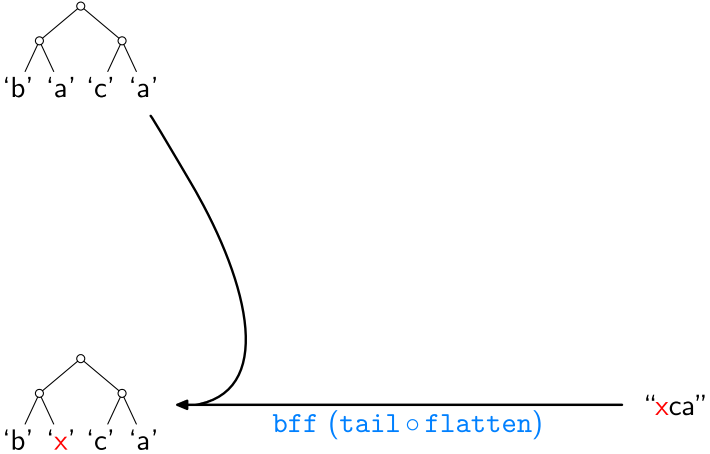
0 → 'b'
1 → 'a'
2 → 'c'
3 → 'a'

0 → 'b'
1 → 'x'
2 → 'c'
3 → 'a'

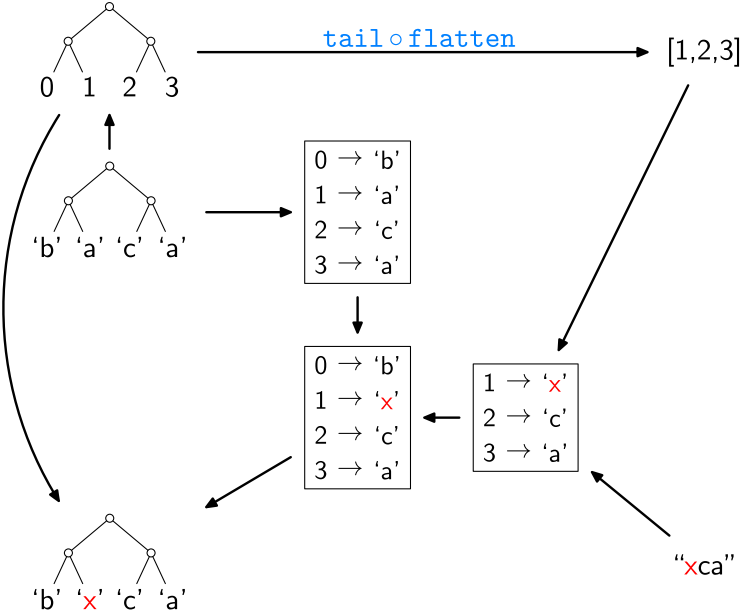
1 → 'x'
2 → 'c'
3 → 'a'

"xca"

The Resulting Bidirectionalisation Scheme by Example



The Resulting Bidirectionalisation Scheme by Example



`tail ∘ flatten`

[1,2,3]

0 → 'b'
1 → 'a'
2 → 'c'
3 → 'a'

0 → 'b'
1 → 'x'
2 → 'c'
3 → 'a'

1 → 'x'
2 → 'c'
3 → 'a'

"xca"

The Implementation (here: lists only, inefficient version)

```
bff get s v' = let n  = (length s) − 1
                   t  = [0..n]
                   g  = zip t s
                   h  = assoc (get t) v'
                   h' = h ++ g
               in  seq h (map (λi → fromJust (lookup i h')) t)

assoc []       []       = []
assoc (i : is) (b : bs) = let m = assoc is bs
                          in  case lookup i m of
                                Nothing          → (i, b) : m
                                Just c | b == c  → m
```

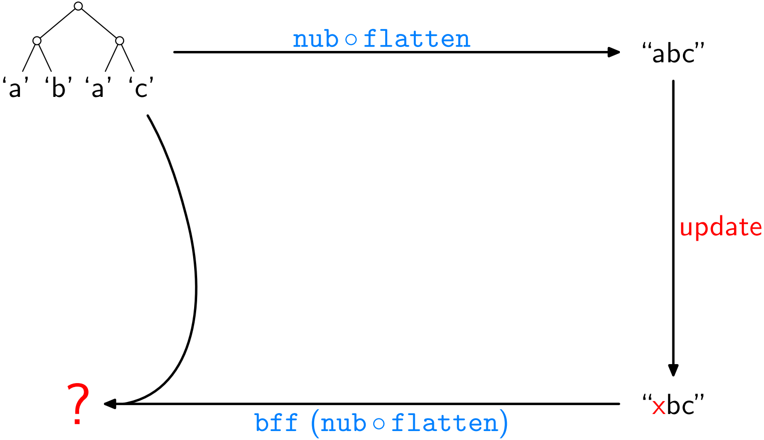
The Implementation (here: lists only, inefficient version)

```
bff get s v′ = let n  = (length s) − 1
                   t  = [0..n]
                   g  = zip t s
                   h  = assoc (get t) v′
                   h′ = h ++ g
               in  seq h (map (λi → fromJust (lookup i h′)) t)

assoc []       []       = []
assoc (i : is) (b : bs) = let m = assoc is bs
                          in  case lookup i m of
                                Nothing          → (i, b) : m
                                Just c | b == c → m
```

- actual code only slightly more elaborate
- online: `http://www-ps.iai.uni-bonn.de/cgi-bin/bff.cgi`

Another Interesting Example



‘a’ ‘b’ ‘a’ ‘c’

$\texttt{nub} \circ \texttt{flatten}$

“abc”

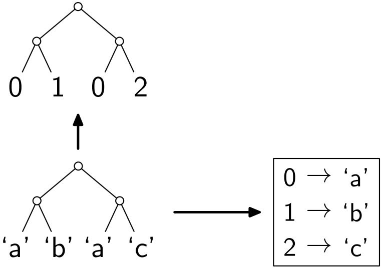
update

“xbc”

?

$\texttt{bff}\ (\texttt{nub} \circ \texttt{flatten})$

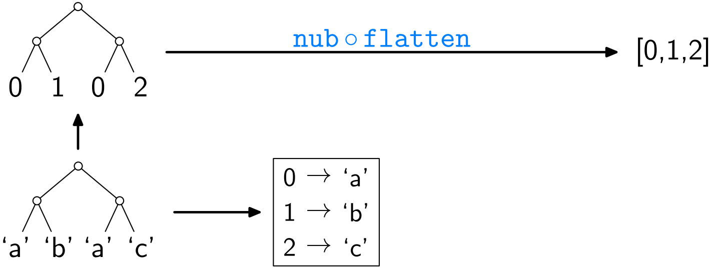
Another Interesting Example



0 → ‘a’
1 → ‘b’
2 → ‘c’

“xbc”

Another Interesting Example



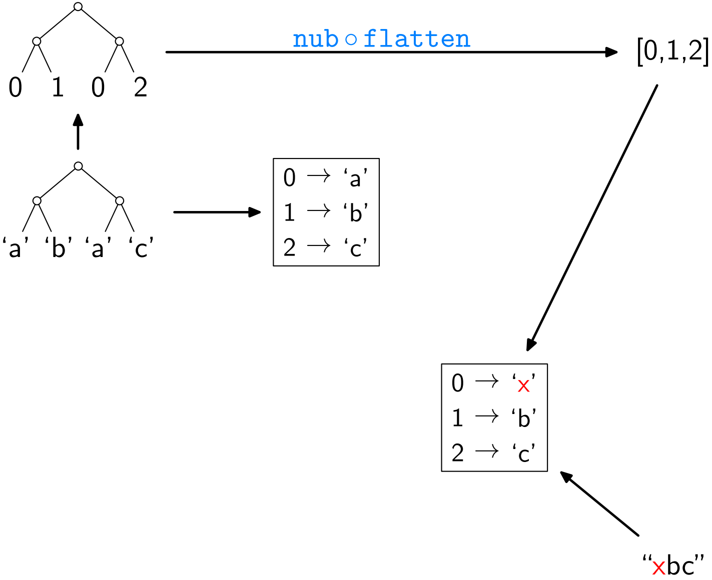
`nub ∘ flatten`

[0,1,2]

0 → 'a'
1 → 'b'
2 → 'c'

"xbc"

Another Interesting Example



`nub` ∘ `flatten`

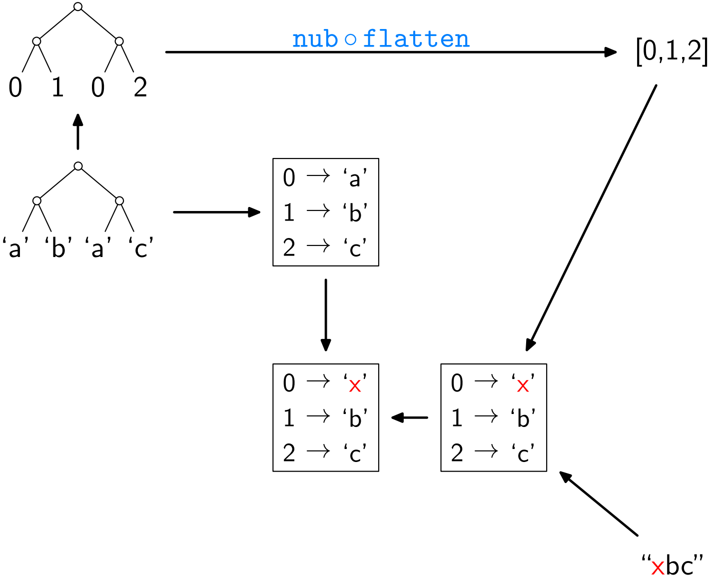
[0,1,2]

0 → 'a'
1 → 'b'
2 → 'c'

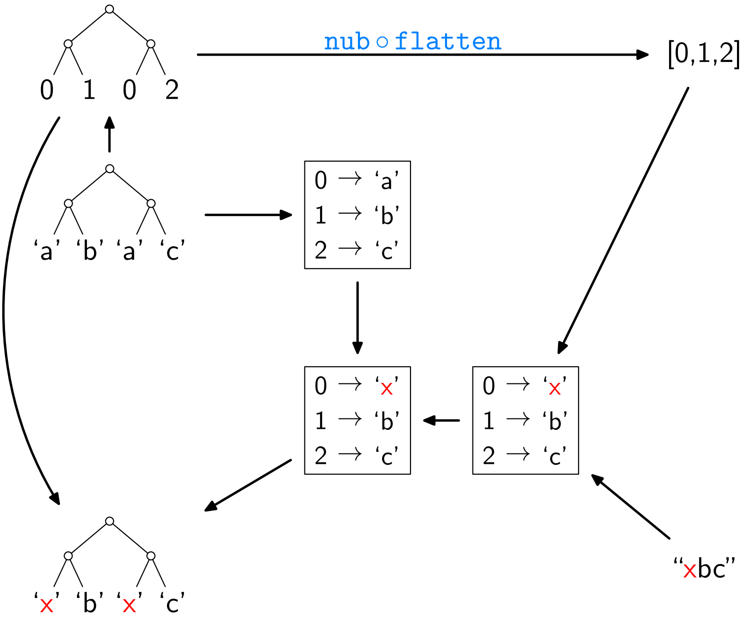
0 → 'x'
1 → 'b'
2 → 'c'

"xbc"

Another Interesting Example



Another Interesting Example



What Else?

[V. 2009]:

- full treatment of equality and ordering constraints
- proofs, using free theorems and equational reasoning
- a datatype-generic account of the whole story

What Else?

[V. 2009]:

- full treatment of equality and ordering constraints
- proofs, using free theorems and equational reasoning
- a datatype-generic account of the whole story

Pros of the approach:

- liberation from syntactic constraints
- very lightweight, easy access to bidirectionality

What Else?

[V. 2009]:

- full treatment of equality and ordering constraints
- proofs, using free theorems and equational reasoning
- a datatype-generic account of the whole story

Pros of the approach:

- liberation from syntactic constraints
- very lightweight, easy access to bidirectionality

Cons of the approach:

- efficiency still leaves room for improvement
- partiality, e.g., rejection of shape-affecting updates

What Else?

[V. 2009]:

- full treatment of equality and ordering constraints
- proofs, using free theorems and equational reasoning
- a datatype-generic account of the whole story

Pros of the approach:

- liberation from syntactic constraints
- very lightweight, easy access to bidirectionality

Cons of the approach:

- efficiency still leaves room for improvement
- partiality, e.g., rejection of shape-affecting updates

[V. et al. 2010]:

- a synthesis of syntactic and semantic bidirectionalisation
- . . . to the benefit of both approaches

References I

F. Bancilhon and N. Spyratos.
Update semantics of relational views.
ACM Transactions on Database Systems, 6(3):557–575, 1981.

J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, and A. Schmitt.
Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem.
ACM Transactions on Programming Languages and Systems, 29(3):17, 2007.

K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi.
Bidirectionalization transformation based on automatic derivation of view complement functions.
In *International Conference on Functional Programming, Proceedings*, pages 47–58. ACM Press, 2007.

References II

J. Voigtländer.
Bidirectionalization for free!
In *Principles of Programming Languages, Proceedings*, pages 165–176. ACM Press, 2009.

J. Voigtländer, Z. Hu, K. Matsuda, and M. Wang.
Combining syntactic and semantic bidirectionalization.
In *International Conference on Functional Programming, Proceedings*, pages 181–192. ACM Press, 2010.

P. Wadler.
Theorems for free!
In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.