Combining Syntactic and Semantic Bidirectionalization

J. Voigtländer[1] Z. Hu[2] K. Matsuda[3] M. Wang[4]

[1]University of Bonn

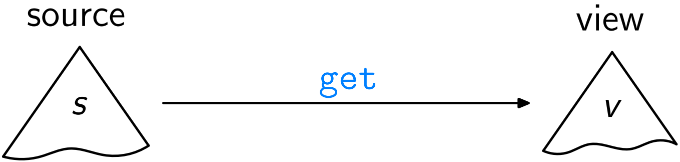
[2]NII Tokyo

[3]Tohoku University

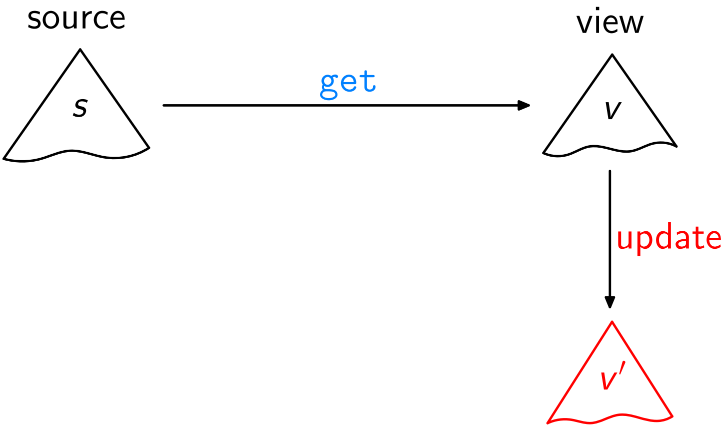
[4]University of Oxford

ICFP'10

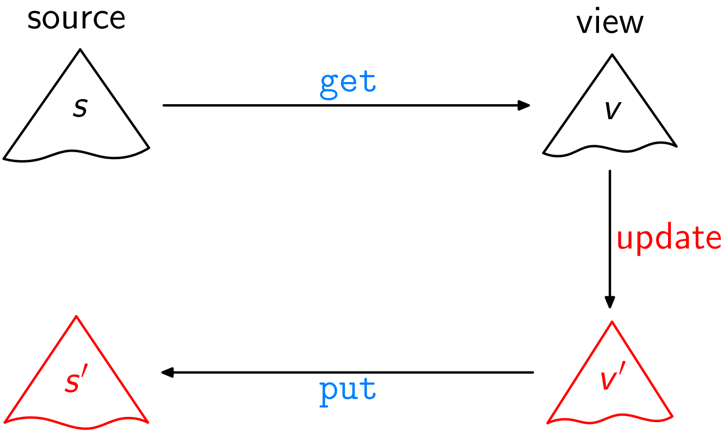
Bidirectional Transformation



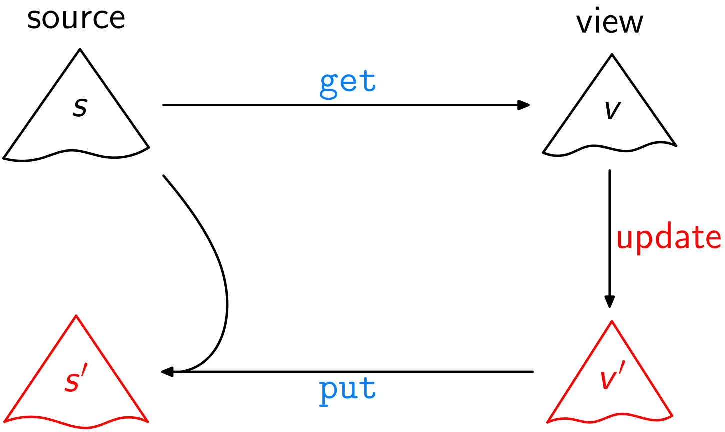
Bidirectional Transformation



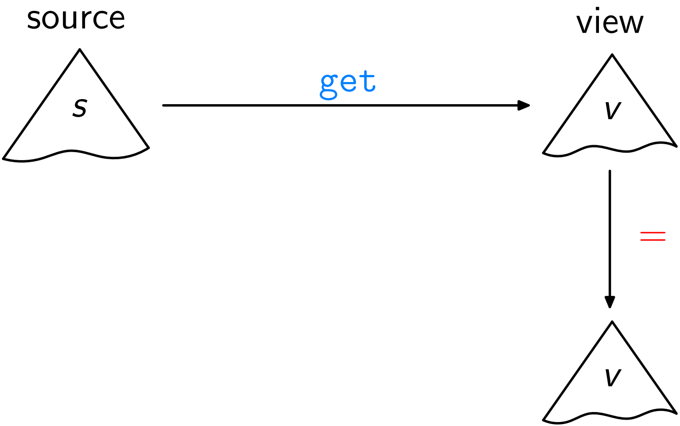
Bidirectional Transformation



Bidirectional Transformation

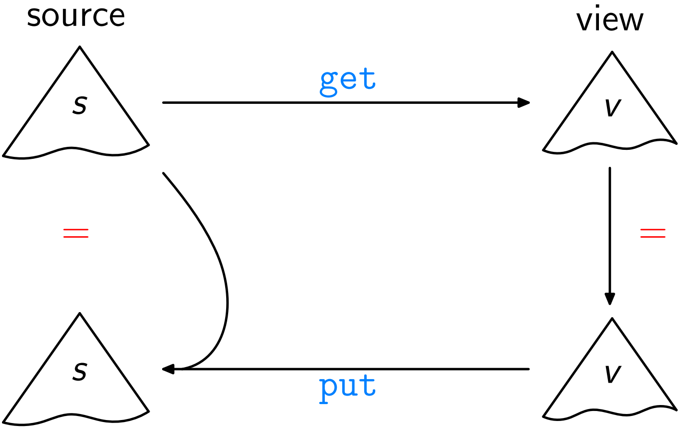


Bidirectional Transformation



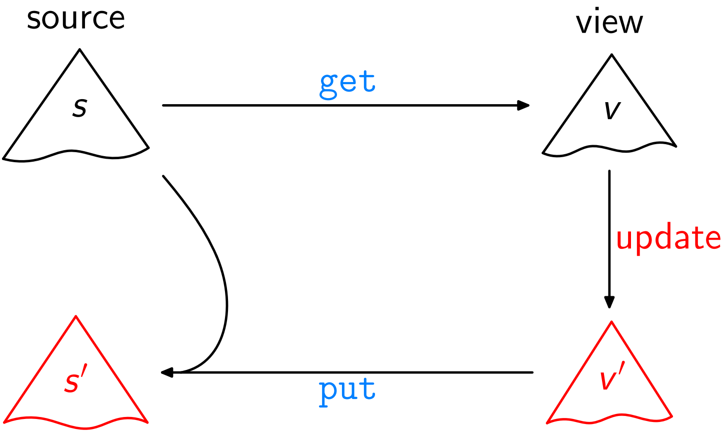
Acceptability / GetPut

Bidirectional Transformation



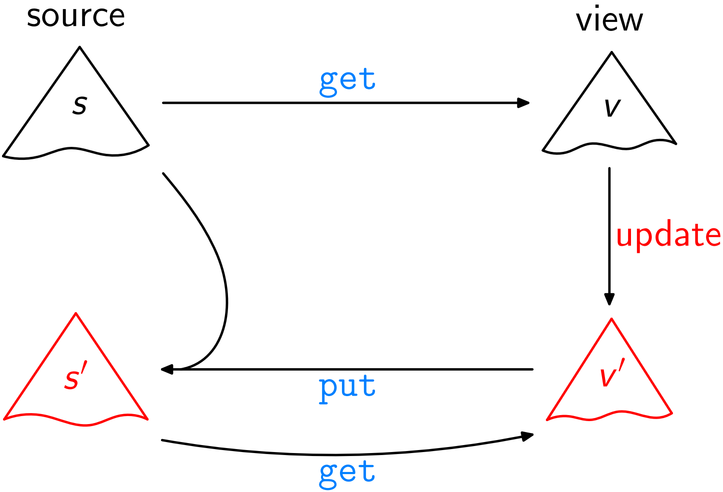
Acceptability / GetPut

Bidirectional Transformation



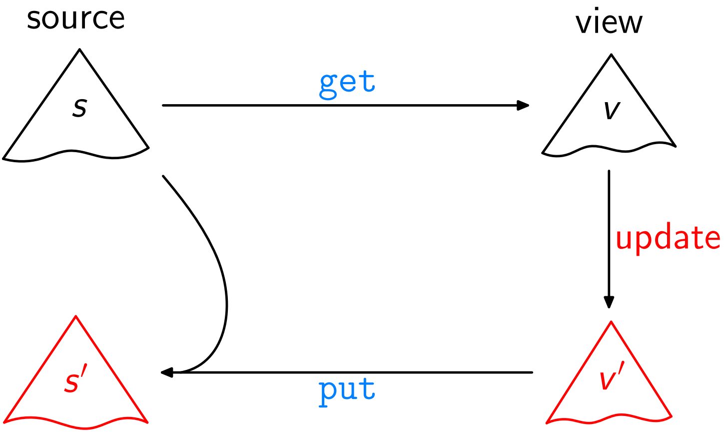
Consistency / PutGet

Bidirectional Transformation

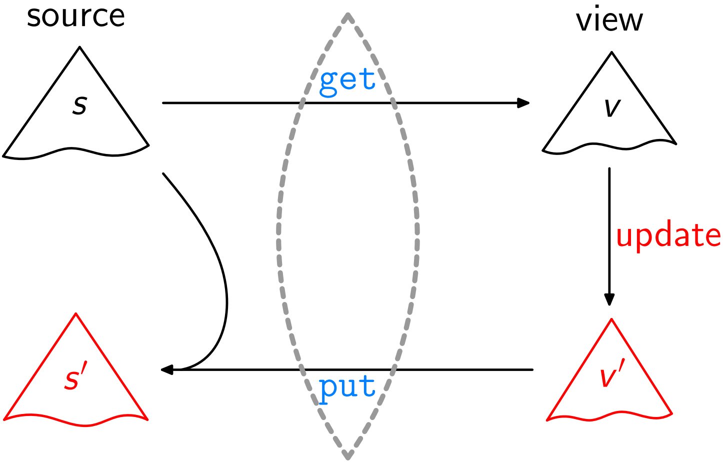


Consistency / PutGet

Bidirectional Transformation



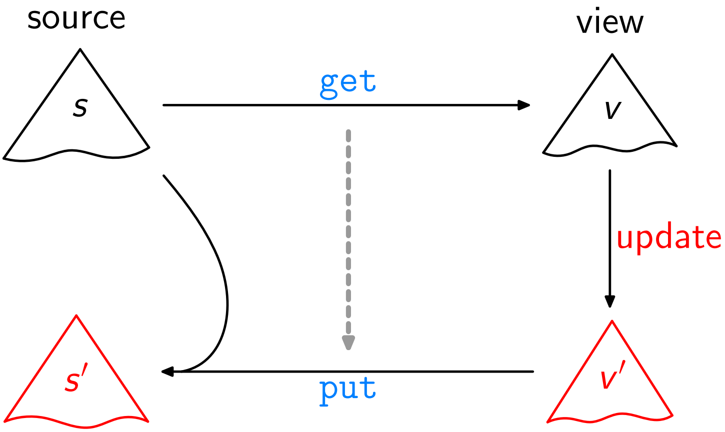
Bidirectional Transformation



Lenses, DSLs

[Foster et al., TOPLAS'07, ...]

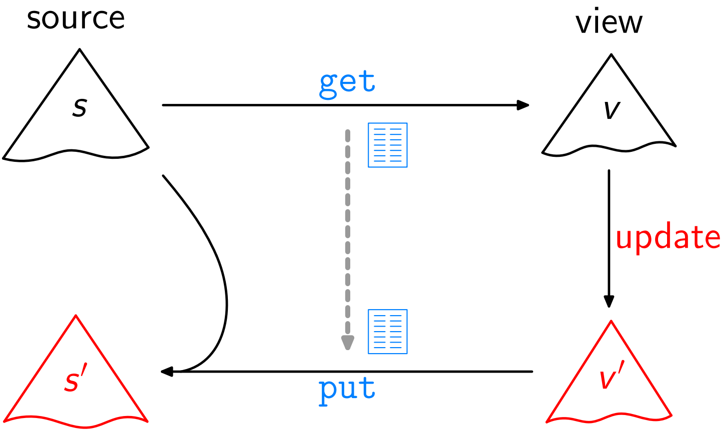
Bidirectional Transformation



Bidirectionalization

[Matsuda et al., ICFP'07]

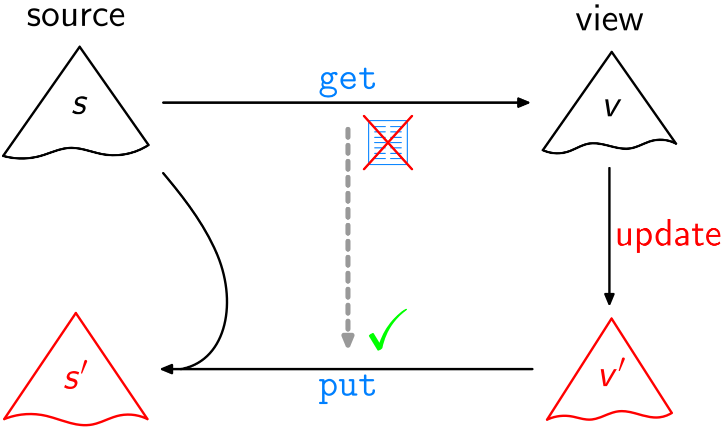
Bidirectional Transformation



Syntactic Bidirectionalization

[Matsuda et al., ICFP'07]

Bidirectional Transformation



Semantic Bidirectionalization

[V., POPL'09]

Semantic Bidirectionalization

Idea: Have higher-order function `bff`[†] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

[†] "<u>B</u>idirectionalization <u>f</u>or <u>f</u>ree!"

Semantic Bidirectionalization

Idea: Have higher-order function `bff`[†] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

“abc” $\xrightarrow{\texttt{tail}}$ “bc”

† “Bidirectionalization for free!”

Semantic Bidirectionalization

Idea: Have higher-order function `bff`† such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

† "Bidirectionalization for free!"

Semantic Bidirectionalization

Idea: Have higher-order function `bff`† such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

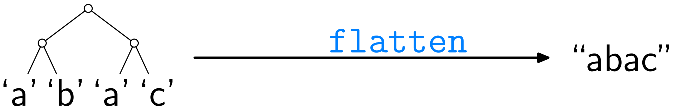
$$
\begin{tikzcd}
\text{“abc”} \arrow[rr, "\texttt{tail}"] \arrow[d, bend left] & & \text{“bc”} \arrow[d, "\text{update}"] \\
\text{“ade”} & & \text{“de”} \arrow[ll, "\texttt{bff tail}"]
\end{tikzcd}
$$

† “Bidirectionalization for free!”

Semantic Bidirectionalization

Idea: Have higher-order function `bff`† such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:



† "Bidirectionalization for free!"

Semantic Bidirectionalization

Idea: Have higher-order function `bff`† such that any `get` and `bff get` satisfy GetPut, PutGet, ...

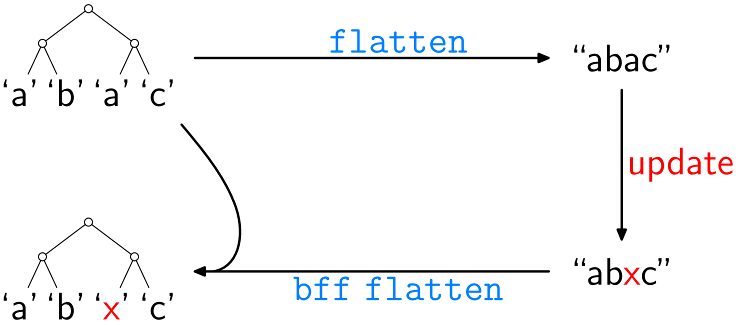
Examples:

† “Bidirectionalization for free!”

Semantic Bidirectionalization

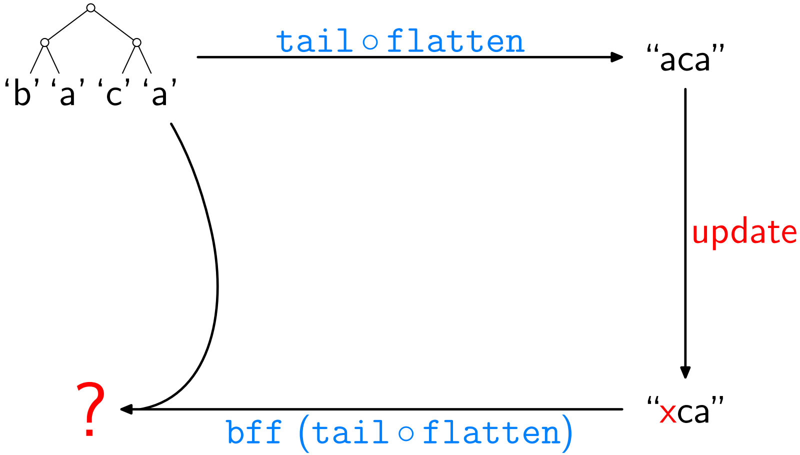
Idea: Have higher-order function `bff`[†] such that any `get` and `bff get` satisfy GetPut, PutGet, ...

Examples:

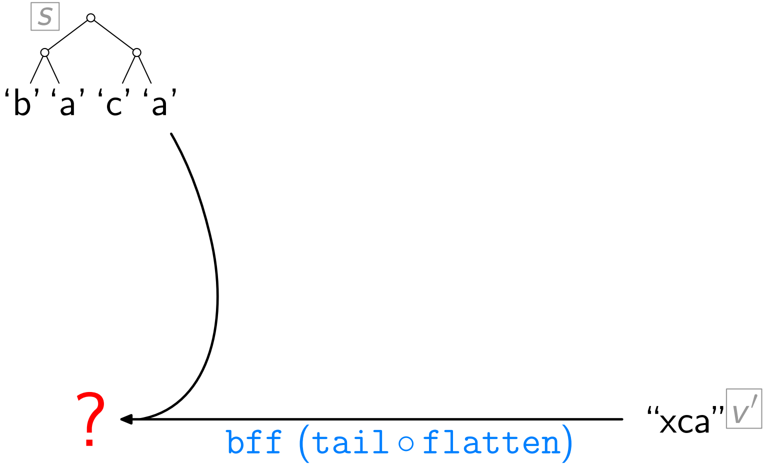


† "Bidirectionalization for free!"

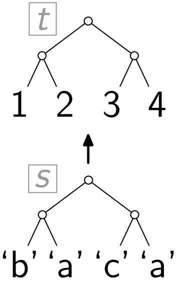
The Semantic Approach by Example



The Semantic Approach by Example



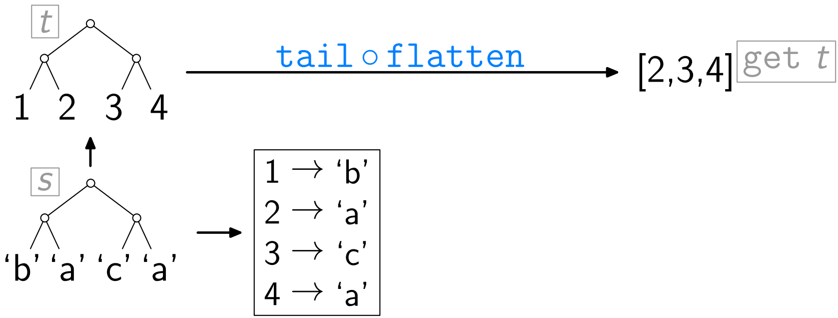
The Semantic Approach by Example

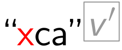


$1 \rightarrow$ 'b'
$2 \rightarrow$ 'a'
$3 \rightarrow$ 'c'
$4 \rightarrow$ 'a'

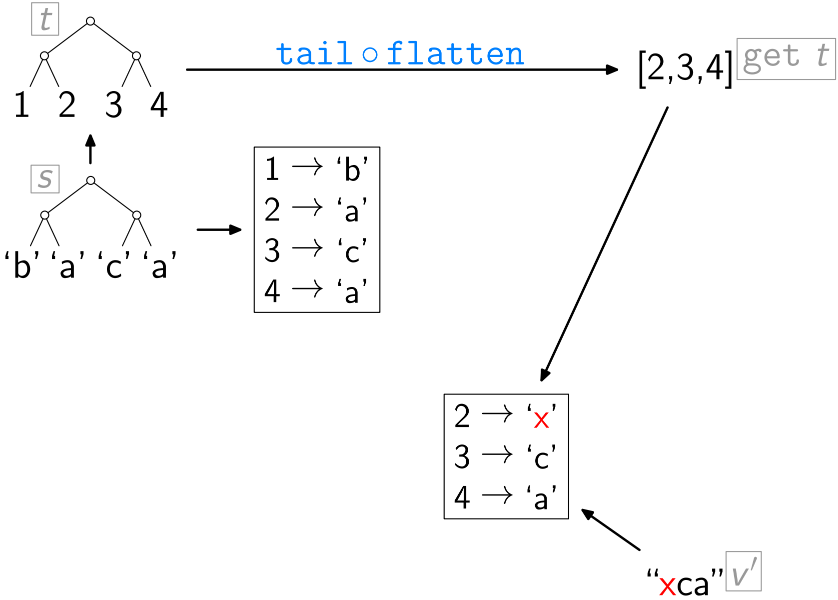
"xca" v'

The Semantic Approach by Example

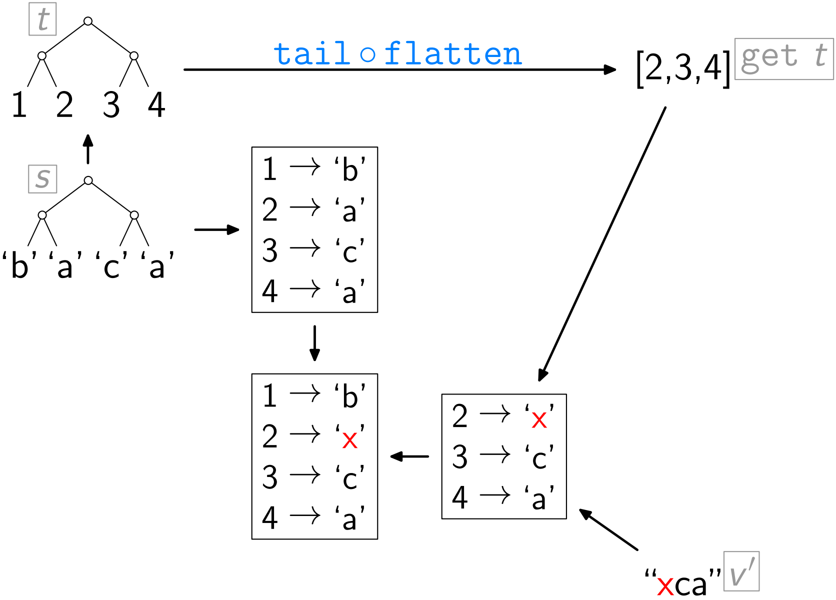




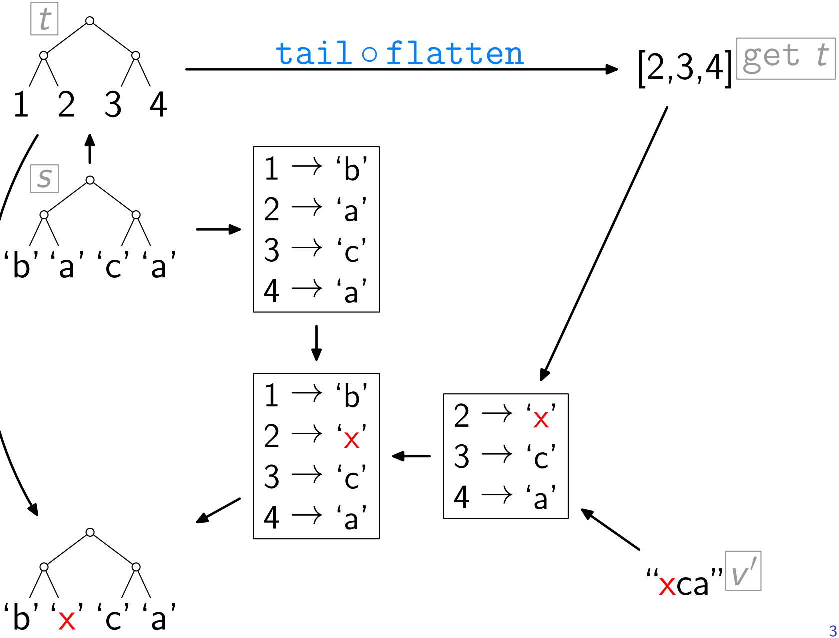
The Semantic Approach by Example



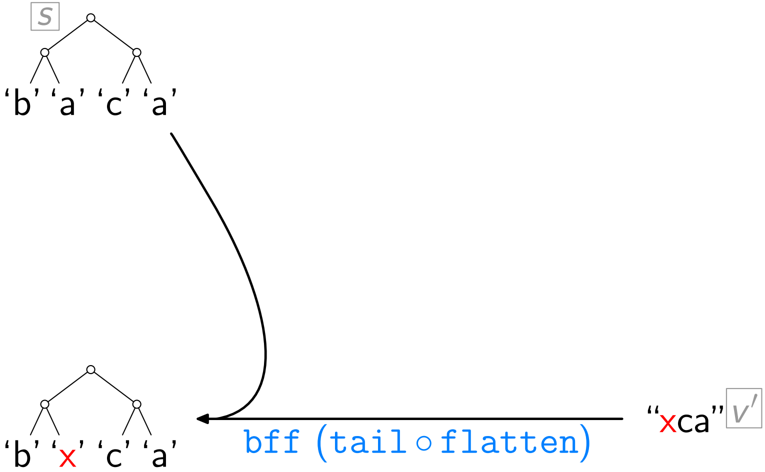
The Semantic Approach by Example



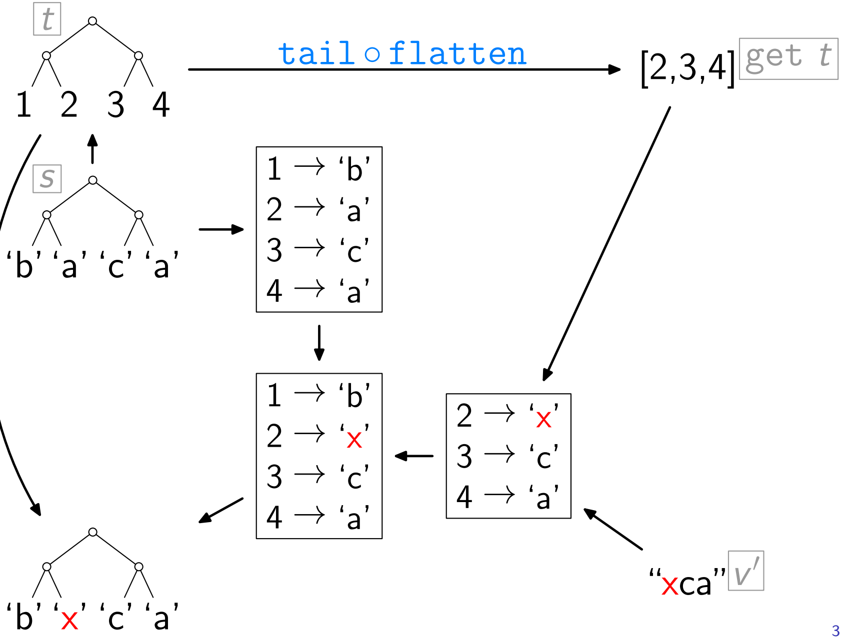
The Semantic Approach by Example



The Semantic Approach by Example



The Semantic Approach by Example



“Status Quo”

[V., POPL’09]:

- very lightweight, easy access to bidirectionality
- proofs by free theorems [Wadler, FPCA’89]
- major problem: rejects shape-changing updates

"Status Quo"

[V., POPL'09]:

- very lightweight, easy access to bidirectionality
- proofs by free theorems [Wadler, FPCA'89]
- major problem: rejects shape-changing updates

[Matsuda et al., ICFP'07]:

- heavily depends on syntactic restraints
- allows (ad-hoc) also shape-changing updates

“Status Quo”

[V., POPL’09]:

- very lightweight, easy access to bidirectionality
- proofs by free theorems [Wadler, FPCA’89]
- major problem: rejects shape-changing updates

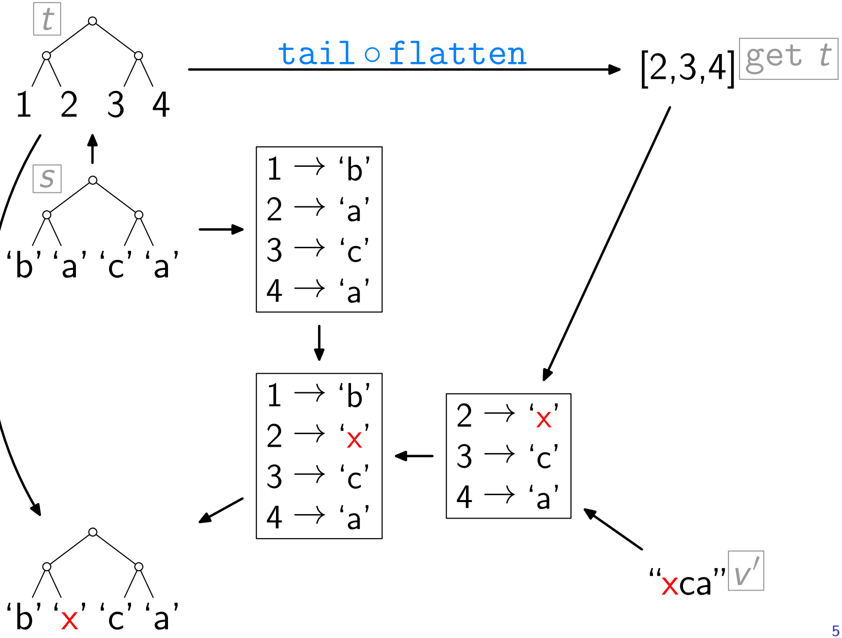
[Matsuda et al., ICFP’07]:

- heavily depends on syntactic restraints
- allows (ad-hoc) also shape-changing updates

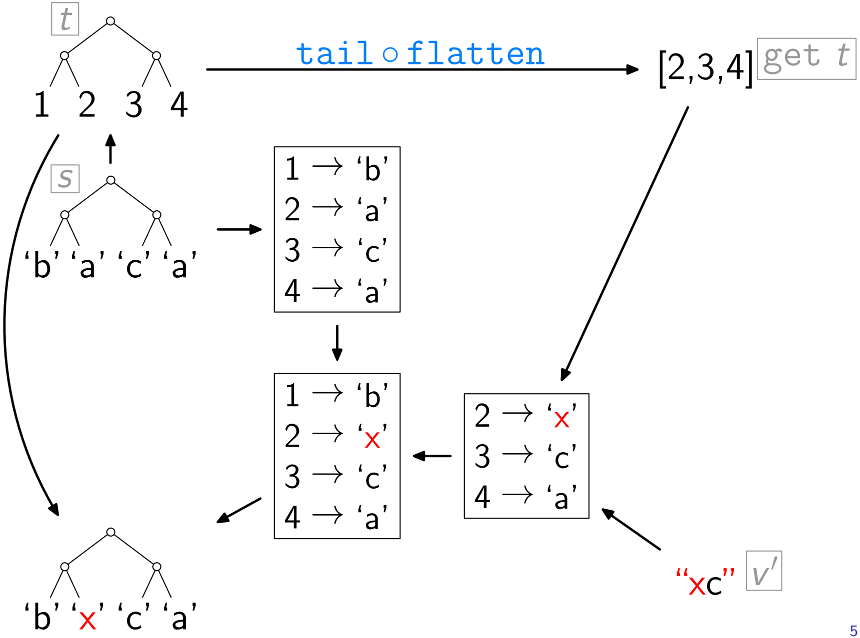
Here:

- synthesis of the two techniques
- inherits limitations in program coverage from both
- strictly better in terms of updatability than either

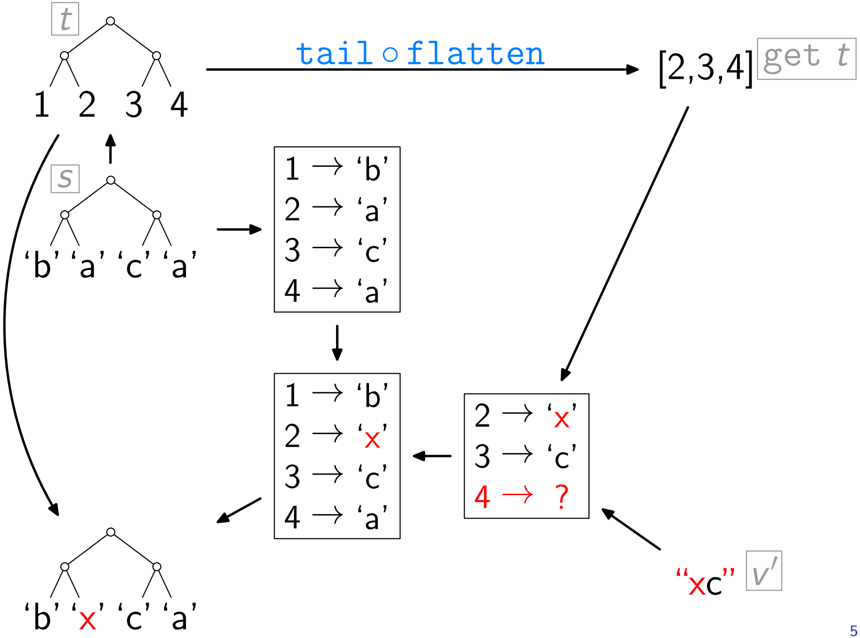
More Shape-Flexibility



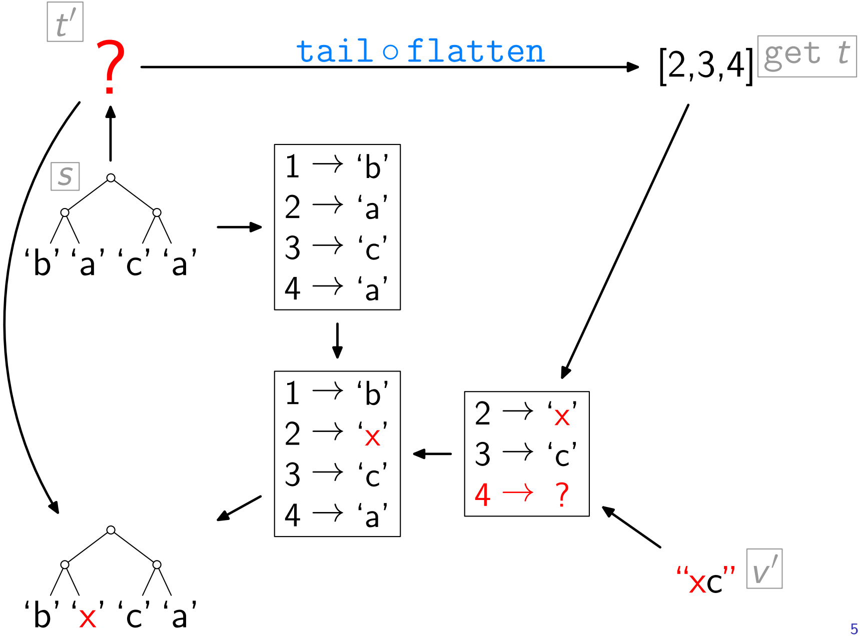
More Shape-Flexibility



More Shape-Flexibility



More Shape-Flexibility



Expectations on $\mathbf{t}'$

Let σ be a function which given a data structure computes a representation of its shape.

Then we want:

1. $\sigma(\texttt{get}\ t') = \sigma(v')$
2. if $\sigma(v') = \sigma(\texttt{get}\ s)$, then $\sigma(t') = \sigma(s)$

Expectations on t′

Let σ be a function which given a data structure computes a representation of its shape.

Then we want:

1. $\sigma(\texttt{get}\ t') = \sigma(v')$
2. if $\sigma(v') = \sigma(\texttt{get}\ s)$, then $\sigma(t') = \sigma(s)$

Key Idea: Abstraction!

Find sget such that:

$$
\begin{tikzcd}
\circ \arrow[r, "\texttt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\
\circ \arrow[r, "\texttt{sget}"'] & \circ
\end{tikzcd}
$$

“Bootstrapping”

For `sget`, find `sput` such that GetPut and PutGet hold:

$$
\begin{tikzcd}
\circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\
\circ & \circ \arrow[l, "\texttt{sput}"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\
\circ \arrow[r, bend right, "\texttt{sget}"'] & \circ \arrow[l, "\texttt{sput}"]
\end{tikzcd}
$$

“Bootstrapping”

For `sget`, find `sput` such that GetPut and PutGet hold:

$$
\begin{tikzcd}
\circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\
\circ & \circ \arrow[l, "\mathtt{sput}"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\
\circ \arrow[r, bend right, "\mathtt{sget}"'] & \circ \arrow[l, "\mathtt{sput}"]
\end{tikzcd}
$$

Then, set t' such that:

$$
\begin{tikzcd}
\sigma(s) \arrow[d, bend left] & \\
\sigma(t') & \sigma(v') \arrow[l, "\mathtt{sput}"']
\end{tikzcd}
$$

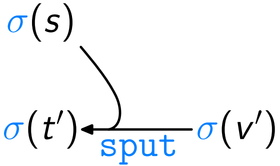
Expectations on t'

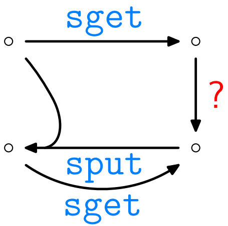
1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

Expectations on t'

1. $\sigma(\texttt{get}\ t') = \sigma(v')$?

From:





Expectations on t'

1. $\sigma(\mathtt{get}\ t') = \sigma(v')$?

From:

$$\begin{tikzcd} \sigma(s) \arrow[d, bend left] & \\ \sigma(t') & \sigma(v') \arrow[l, "\mathtt{sput}"'] \end{tikzcd}$$

$$\begin{tikzcd} \circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\ \circ \arrow[r, bend right, "\mathtt{sget}"'] & \circ \arrow[l, "\mathtt{sput}"'] \end{tikzcd}$$

follows:

$$\begin{tikzcd} \sigma(s) \arrow[r, "\mathtt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\ \sigma(t') \arrow[r, bend right, "\mathtt{sget}"'] & \sigma(v') \arrow[l, "\mathtt{sput}"'] \end{tikzcd}$$

Expectations on t'

1. $\sigma(\mathtt{get}\ t') = \sigma(v')$?

From:

$$\begin{tikzcd}
\sigma(s) \arrow[r, "\mathtt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\
\sigma(t') \arrow[r, bend right, "\mathtt{sget}"'] & \sigma(v') \arrow[l, "\mathtt{sput}"']
\end{tikzcd}$$

Expectations on t'

1. $\sigma(\mathtt{get}\ t') = \sigma(v')$?

From:

$$\begin{tikzcd}
\sigma(s) \arrow[r, "\mathtt{sget}"] & \circ \arrow[d, "?"] \\
\sigma(t') \arrow[r, bend right, "\mathtt{sget}"'] & \sigma(v') \arrow[l, "\mathtt{sput}"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
t' \arrow[r, "\mathtt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\
\circ \arrow[r, "\mathtt{sget}"'] & \circ
\end{tikzcd}$$

follows: $\sigma(\mathtt{get}\ t') = \sigma(v')$.

Expectations on t'

2. if $\sigma(v') = \sigma(\texttt{get}\ s)$, then $\sigma(t') = \sigma(s)$?

Expectations on t'

2. if $\sigma(v') = \sigma(\mathtt{get}\ s)$, then $\sigma(t') = \sigma(s)$?

From:

$$\begin{tikzcd} \sigma(s) \arrow[d, bend left] & \\ \sigma(t') & \sigma(v') \arrow[l, "\mathtt{sput}"'] \end{tikzcd} \qquad \begin{tikzcd} \circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\ \circ & \circ \arrow[l, "\mathtt{sput}"'] \end{tikzcd}$$

Expectations on t'

2. if $\sigma(v') = \sigma(\texttt{get}\ s)$, then $\sigma(t') = \sigma(s)$?

From:

$$\begin{tikzcd}
\sigma(s) \arrow[d, bend left] & \\
\sigma(t') & \sigma(v') \arrow[l, "\texttt{sput}"']
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left, phantom, ""] \arrow[d, bend left] \arrow[d, phantom, "\color{red}{=}"', shift right=3ex] & \circ \arrow[d] \arrow[d, phantom, "\color{red}{=}", shift left=3ex] \\
\circ & \circ \arrow[l, "\texttt{sput}"]
\end{tikzcd}$$

follows:

$$\begin{tikzcd}
\sigma(s) \arrow[r, "\texttt{sget}"] \arrow[d, bend left] \arrow[d, phantom, "\color{red}{=}"', shift right=3ex] & \circ \arrow[d] \arrow[d, phantom, "\color{red}{=}", shift left=3ex] \\
\sigma(t') & \sigma(v') \arrow[l, "\texttt{sput}"]
\end{tikzcd}$$

Expectations on t'

2. if $\sigma(v') = \sigma(\texttt{get}\ s)$, then $\sigma(t') = \sigma(s)$?

From:

$$
\begin{tikzcd}
\sigma(s) \arrow[r, "\texttt{sget}"] \arrow[d, phantom, "="] & \circ \arrow[d] \arrow[d, phantom, shift left=4ex, "="] \\
\sigma(t') & \sigma(v') \arrow[l, "\texttt{sput}"]
\end{tikzcd}
$$

Expectations on t'

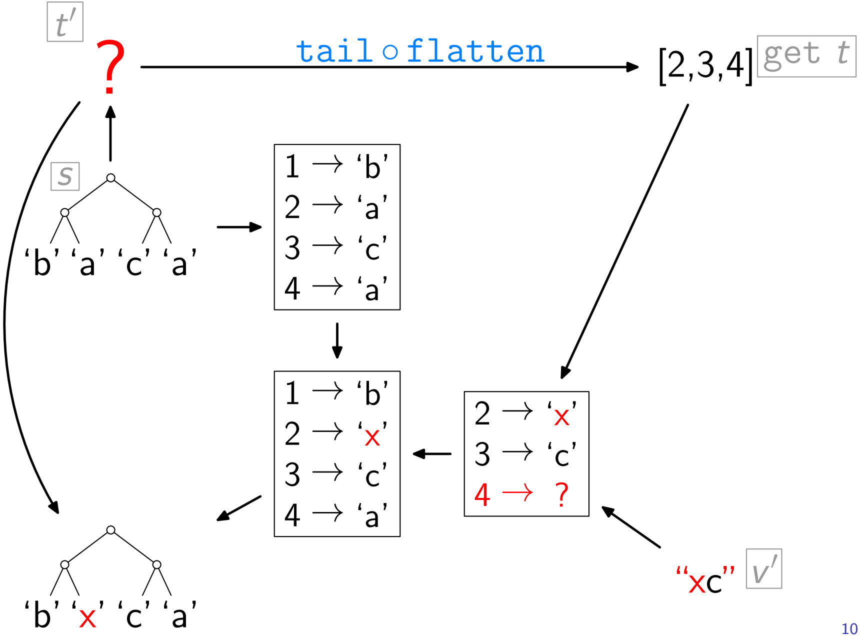
2. if $\sigma(v') = \sigma(\mathtt{get}\ s)$, then $\sigma(t') = \sigma(s)$?

From:

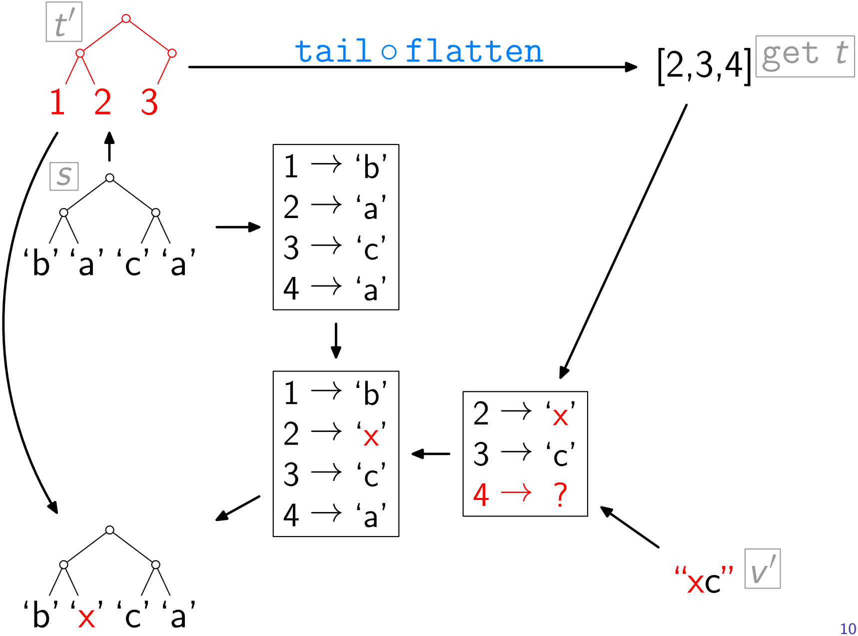
$$
\begin{tikzcd}
\sigma(s) \arrow[r, "\mathtt{sget}"] \arrow[d, phantom, "="] & \circ \arrow[d, "="] \\
\sigma(t') & \sigma(v') \arrow[l, "\mathtt{sput}"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
s \arrow[r, "\mathtt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\
\circ \arrow[r, "\mathtt{sget}"'] & \circ
\end{tikzcd}
$$

follows that if $\sigma(v') = \sigma(\mathtt{get}\ s)$, then $\sigma(t') = \sigma(s)$.

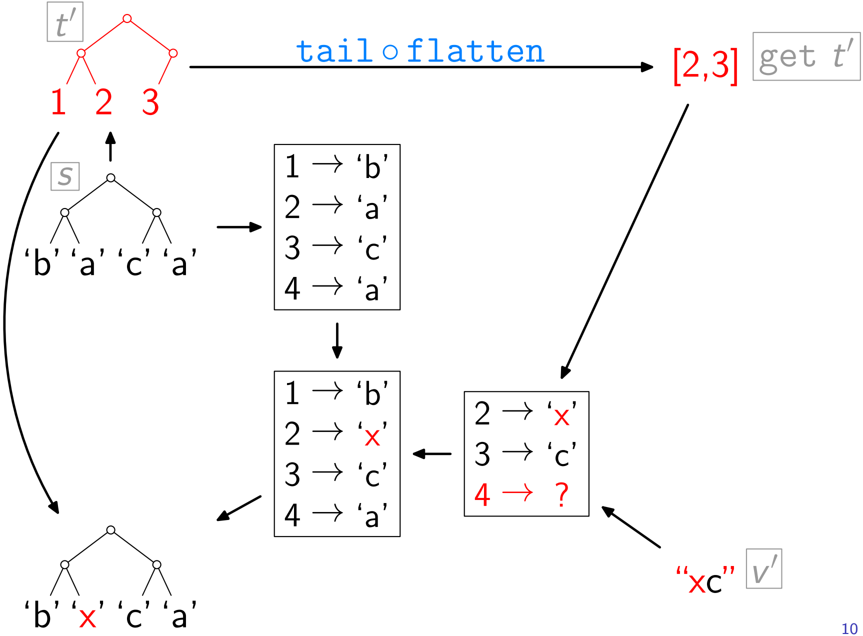
More Shape-Flexibility



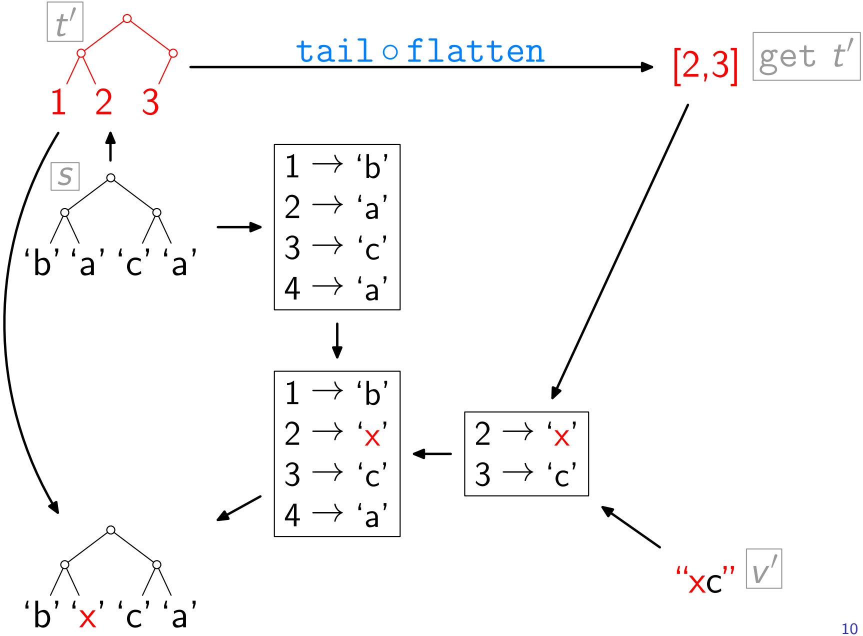
More Shape-Flexibility



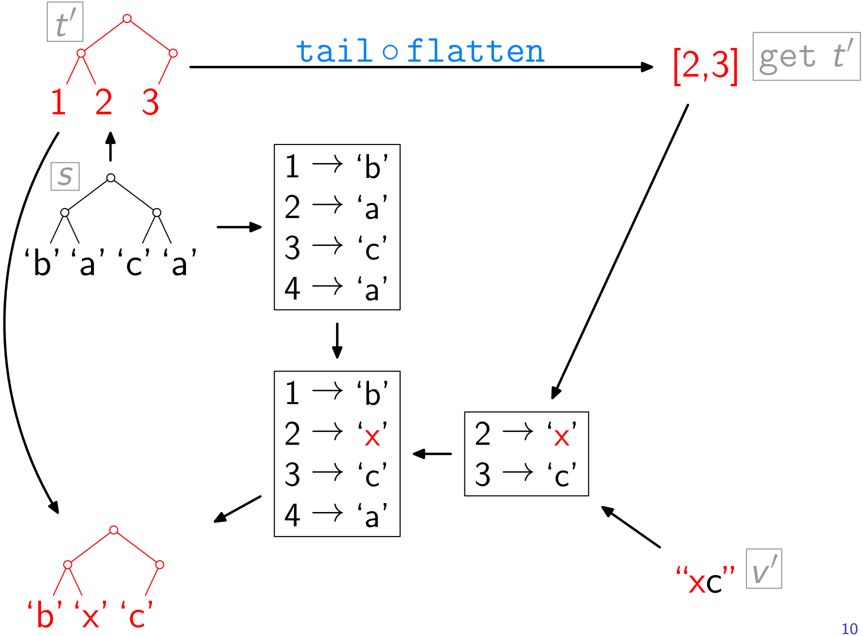
More Shape-Flexibility



More Shape-Flexibility



More Shape-Flexibility



Essential Ingredients

The crucial point is to find `sget` with:

$$\begin{tikzcd} \circ \arrow[r, "\texttt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\ \circ \arrow[r, "\texttt{sget}"'] & \circ \end{tikzcd}$$

Essential Ingredients

The crucial point is to find sget with:

$$\begin{tikzcd}
\circ \arrow[r, "\mathtt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\
\circ \arrow[r, "\mathtt{sget}"'] & \circ
\end{tikzcd}$$

as well as sput such that:

$$\begin{tikzcd}
\circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\
\circ & \circ \arrow[l, "\mathtt{sput}"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "\mathtt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\
\circ \arrow[r, bend right, "\mathtt{sget}"'] & \circ \arrow[l, "\mathtt{sput}"']
\end{tikzcd}$$

Essential Ingredients

The crucial point is to find sget with:

$$
\begin{tikzcd}
\circ \arrow[r, "\texttt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\
\circ \arrow[r, "\texttt{sget}"'] & \circ
\end{tikzcd}
$$

syntactic abstraction

as well as sput such that:

$$
\begin{tikzcd}
\circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\
\circ & \circ \arrow[l, "\texttt{sput}"]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
\circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\
\circ \arrow[r, bend right, "\texttt{sget}"'] & \circ \arrow[l, "\texttt{sput}"]
\end{tikzcd}
$$

Essential Ingredients

The crucial point is to find `sget` with:

$$\begin{tikzcd} \circ \arrow[r, "\texttt{get}"] \arrow[d, "\sigma"'] & \circ \arrow[d, "\sigma"] \\ \circ \arrow[r, "\texttt{sget}"'] & \circ \end{tikzcd}$$

syntactic abstraction

as well as `sput` such that:

$$\begin{tikzcd} \circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left, "="'] & \circ \arrow[d, "="] \\ \circ & \circ \arrow[l, "\texttt{sput}"] \end{tikzcd} \qquad \begin{tikzcd} \circ \arrow[r, "\texttt{sget}"] \arrow[d, bend left] & \circ \arrow[d, "?"] \\ \circ \arrow[r, bend right, "\texttt{sget}"'] & \circ \arrow[l, "\texttt{sput}"] \end{tikzcd}$$

[M. et al., ICFP'07]

The Benefits of Abstraction

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] && = [\,] \\
&\texttt{get}\ [x] && = [\,] \\
&\texttt{get}\ (x : y : zs) && = y : (\texttt{get}\ zs)
\end{aligned}
$$

The Benefits of Abstraction

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] && = [\,] \\
&\texttt{get}\ [x] && = [\,] \\
&\texttt{get}\ (x : y : zs) && = y : (\texttt{get}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] && = \mathsf{C}_1 \\
&\texttt{compl}\ [x] && = \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) && = \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
$$

The Benefits of Abstraction

$$
\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha]\\
&\texttt{get}\ [\,] && = [\,]\\
&\texttt{get}\ [x] && = [\,]\\
&\texttt{get}\ (x : y : zs) && = y : (\texttt{get}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{compl}\ [\,] && = \mathsf{C}_1\\
&\texttt{compl}\ [x] && = \mathsf{C}_2\ x\\
&\texttt{compl}\ (x : y : zs) && = \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}
$$

$$\Downarrow$$

$$
\begin{aligned}
&\texttt{put}\ [\,] && [\,] && = [\,]\\
&\texttt{put}\ [x] && [\,] && = [x]\\
&\texttt{put}\ (x : y : zs) && (y' : v') && = \cdots
\end{aligned}
$$

The Benefits of Abstraction

$$\begin{aligned}
&\texttt{get} :: [\alpha] \to [\alpha] \\
&\texttt{get}\ [\,] &&= [\,] \\
&\texttt{get}\ [x] &&= [\,] \\
&\texttt{get}\ (x : y : zs) &&= y : (\texttt{get}\ zs)
\end{aligned}
\quad \Longrightarrow \quad
\begin{aligned}
&\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} \\
&\texttt{sget}\ 0 &&= 0 \\
&\texttt{sget}\ 1 &&= 0 \\
&\texttt{sget}\ (n+2) &&= 1 + (\texttt{sget}\ n)
\end{aligned}$$

$$\Downarrow$$

$$\begin{aligned}
&\texttt{compl}\ [\,] &&= \mathsf{C}_1 \\
&\texttt{compl}\ [x] &&= \mathsf{C}_2\ x \\
&\texttt{compl}\ (x : y : zs) &&= \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{aligned}$$

$$\Downarrow$$

$$\begin{aligned}
&\texttt{put}\ [\,] && [\,] &&= [\,] \\
&\texttt{put}\ [x] && [\,] &&= [x] \\
&\texttt{put}\ (x : y : zs) && (y' : v') &&= \cdots
\end{aligned}$$

The Benefits of Abstraction

$$\begin{array}{lll}
\texttt{get} :: [\alpha] \to [\alpha] & & \\
\texttt{get}\ [\,] & = [\,] & \\
\texttt{get}\ [x] & = [\,] & \\
\texttt{get}\ (x : y : zs) & = y : (\texttt{get}\ zs) &
\end{array}
\quad \Longrightarrow \quad
\begin{array}{ll}
\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} & \\
\texttt{sget}\ 0 & = 0 \\
\texttt{sget}\ 1 & = 0 \\
\texttt{sget}\ (n+2) & = 1 + (\texttt{sget}\ n)
\end{array}$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$\begin{array}{ll}
\texttt{compl}\ [\,] & = \mathsf{C}_1 \\
\texttt{compl}\ [x] & = \mathsf{C}_2\ x \\
\texttt{compl}\ (x : y : zs) & = \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{array}
\qquad
\begin{array}{ll}
\texttt{compl}\ 0 & = \mathsf{C}_1 \\
\texttt{compl}\ 1 & = \mathsf{C}_2 \\
\texttt{compl}\ (n+2) & = \texttt{compl}\ n
\end{array}$$

$$\Downarrow$$

$$\begin{array}{lll}
\texttt{put}\ [\,] & [\,] & = [\,] \\
\texttt{put}\ [x] & [\,] & = [x] \\
\texttt{put}\ (x : y : zs) & (y' : v') & = \cdots
\end{array}$$

The Benefits of Abstraction

$$\begin{array}{l}
\texttt{get} :: [\alpha] \to [\alpha] \\
\texttt{get}\ [\,] \qquad\qquad\quad = [\,] \\
\texttt{get}\ [x] \qquad\qquad\quad = [\,] \\
\texttt{get}\ (x : y : zs) = y : (\texttt{get}\ zs)
\end{array}
\quad\Longrightarrow\quad
\begin{array}{l}
\texttt{sget} :: \mathsf{Int} \to \mathsf{Int} \\
\texttt{sget}\ 0 \qquad\quad = 0 \\
\texttt{sget}\ 1 \qquad\quad = 0 \\
\texttt{sget}\ (n+2) = 1 + (\texttt{sget}\ n)
\end{array}$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$\begin{array}{l}
\texttt{compl}\ [\,] \qquad\qquad\quad = \mathsf{C}_1 \\
\texttt{compl}\ [x] \qquad\qquad\quad = \mathsf{C}_2\ x \\
\texttt{compl}\ (x : y : zs) = \mathsf{C}_3\ x\ (\texttt{compl}\ zs)
\end{array}
\qquad
\begin{array}{l}
\texttt{compl}\ 0 \qquad\quad = \mathsf{C}_1 \\
\texttt{compl}\ 1 \qquad\quad = \mathsf{C}_2 \\
\texttt{compl}\ (n+2) = \texttt{compl}\ n
\end{array}$$

$$\Downarrow \qquad\qquad\qquad\qquad \Downarrow$$

$$\begin{array}{l}
\texttt{put}\ [\,] \qquad\qquad\quad [\,] \qquad\quad = [\,] \\
\texttt{put}\ [x] \qquad\qquad\quad [\,] \qquad\quad = [x] \\
\texttt{put}\ (x : y : zs)\ (y' : v') = \cdots
\end{array}
\qquad
\begin{array}{l}
\texttt{sput}\ 0 \qquad\quad 0 \qquad\qquad = 0 \\
\texttt{sput}\ 1 \qquad\quad 0 \qquad\qquad = 1 \\
\texttt{sput}\ (n+2)\ 0 \qquad\qquad = \cdots \\
\texttt{sput}\ n \qquad\quad (v'+1) = \cdots
\end{array}$$

Taking Stock

- Semantic Approach:
  - lightweight, “as a library”
  - essential role: polymorphic function types

Taking Stock

- Semantic Approach:
  - lightweight, “as a library”
  - essential role: polymorphic function types
- Syntactic Approach:
  - classical program transformation
  - “constant-complement” [Banc. & Sp., TODS’81]

Taking Stock

- Semantic Approach:
  - lightweight, "as a library"
  - essential role: polymorphic function types
- Syntactic Approach:
  - classical program transformation
  - "constant-complement" [Banc. & Sp., TODS'81]
- Combination per "Separation of Concerns":
  - separate data into shape and content
  - treat shape via syntactic approach
  - treat content via semantic approach

Looking Further

- Try it out: link to implementation in the paper!

Looking Further

- Try it out: link to implementation in the paper!
- Side effect: syntactic applicability improved (by using additional program transformations)

Looking Further

- Try it out: link to implementation in the paper!
- Side effect: syntactic applicability improved (by using additional program transformations)
- Parametrization via “bias” and default values

Looking Further

- Try it out: link to implementation in the paper!
- Side effect: syntactic applicability improved
  (by using additional program transformations)
- Parametrization via “bias” and default values
- Efficiency: (still) rather bad

Looking Further

- Try it out: link to implementation in the paper!
- Side effect: syntactic applicability improved
  (by using additional program transformations)
- Parametrization via "bias" and default values
- Efficiency: (still) rather bad
- (More) future work: general types, type classes

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.

References II

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 III

- J. Voigtländer.
  Bidirectionalization for free!
  In *Principles of Programming Languages, Proceedings*, pages 165–176. ACM Press, 2009.
- P. Wadler.
  Theorems for free!
  In *Functional Programming Languages and Computer Architecture, Proceedings*, pages 347–359. ACM Press, 1989.