Complement-Based Bidirectionalization

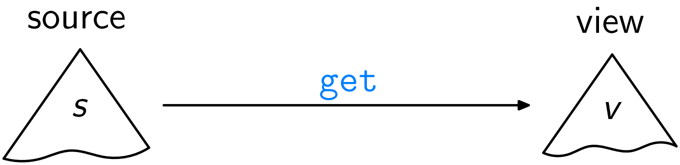
J. Voigtländer

University of Bonn

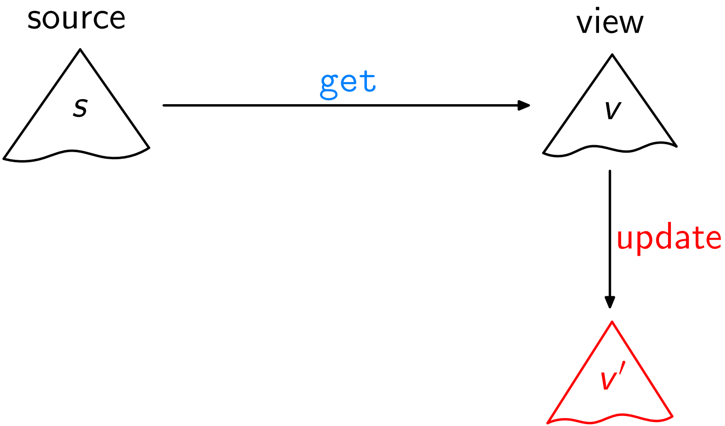
Dagstuhl Seminar “bx”

January 17th, 2011

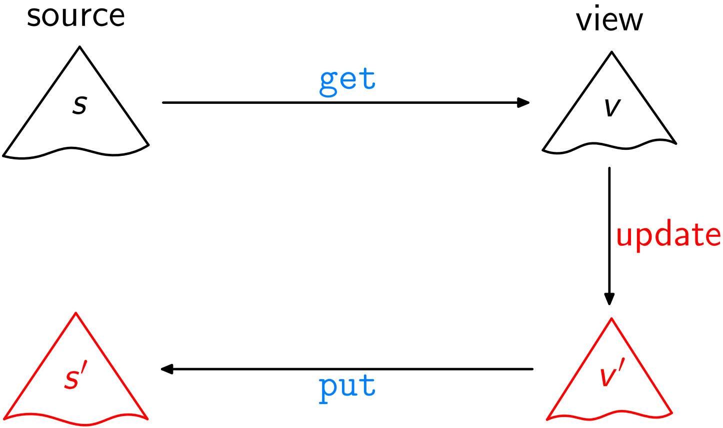
Bidirectional Transformation



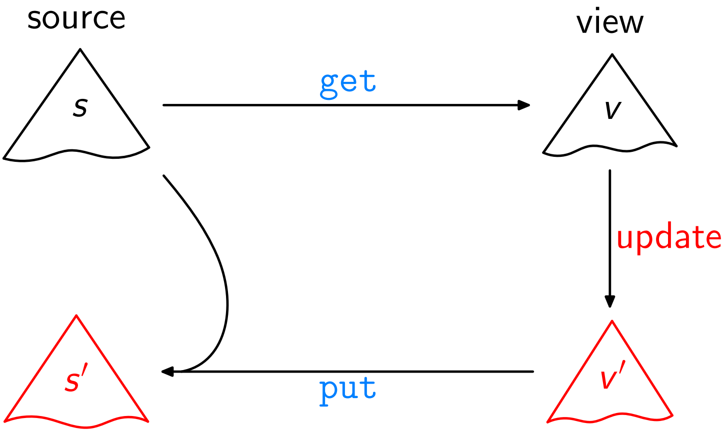
Bidirectional Transformation



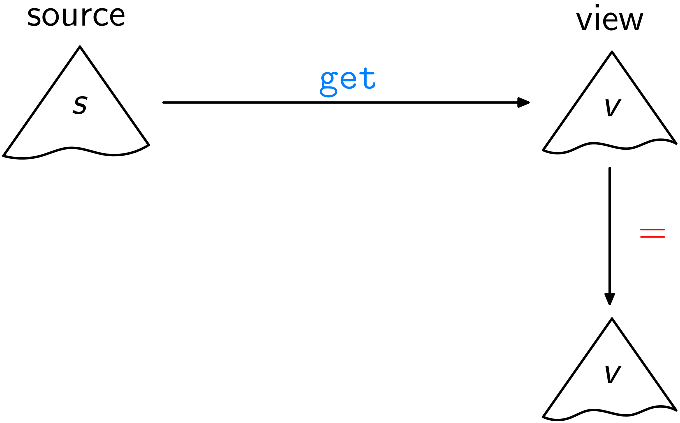
Bidirectional Transformation



Bidirectional Transformation

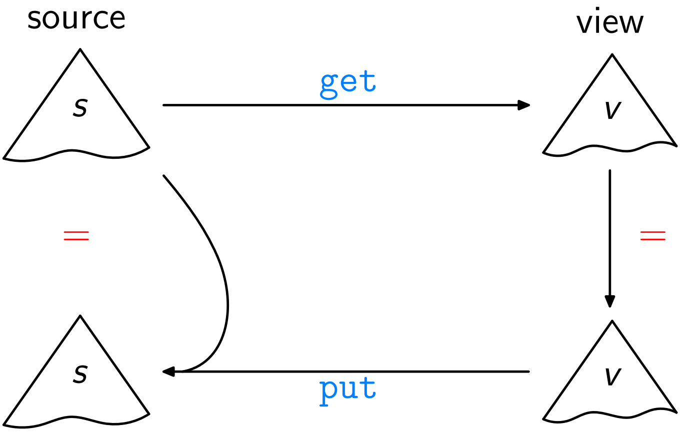


Bidirectional Transformation



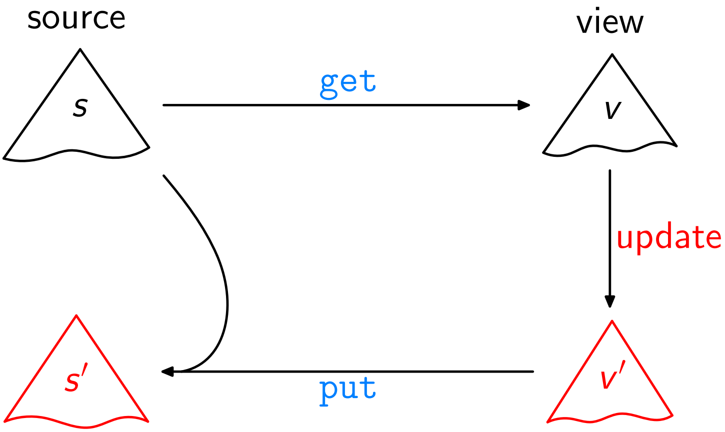
Acceptability / GetPut

Bidirectional Transformation



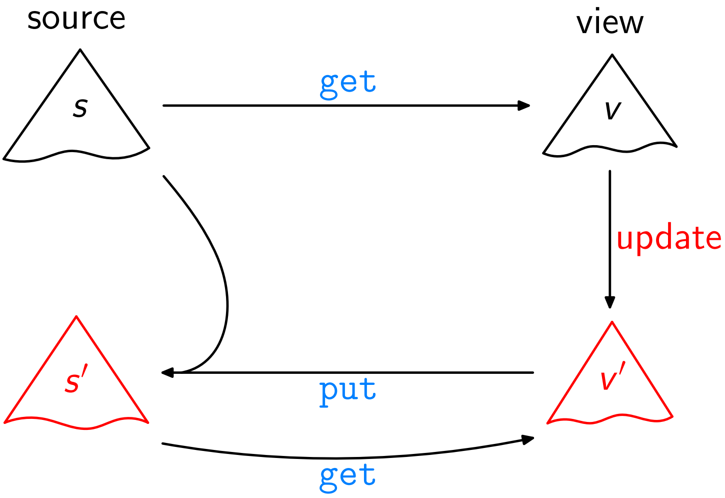
Acceptability / GetPut

Bidirectional Transformation



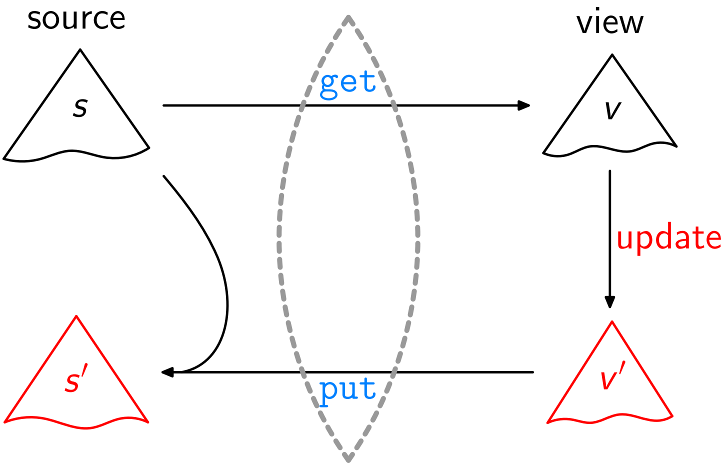
Consistency / PutGet

Bidirectional Transformation

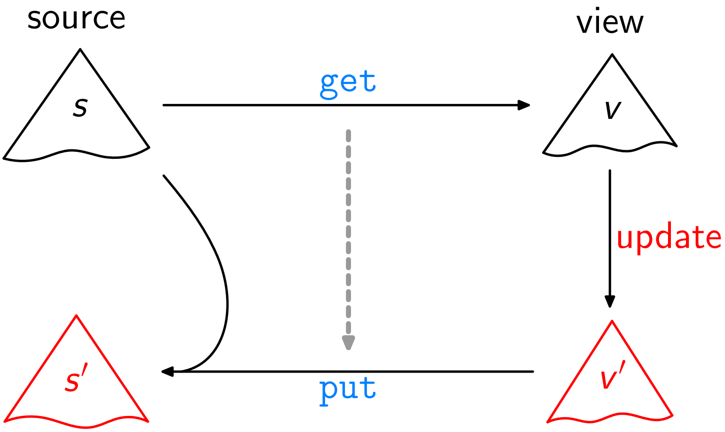


Consistency / PutGet

Bidirectional Transformation

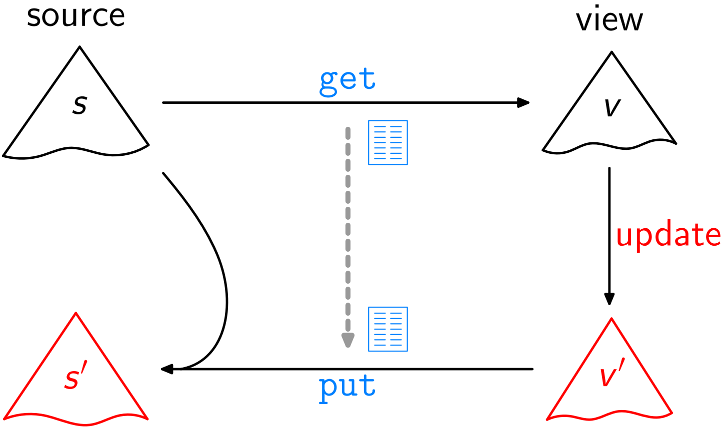


Bidirectional Transformation



Bidirectionalization

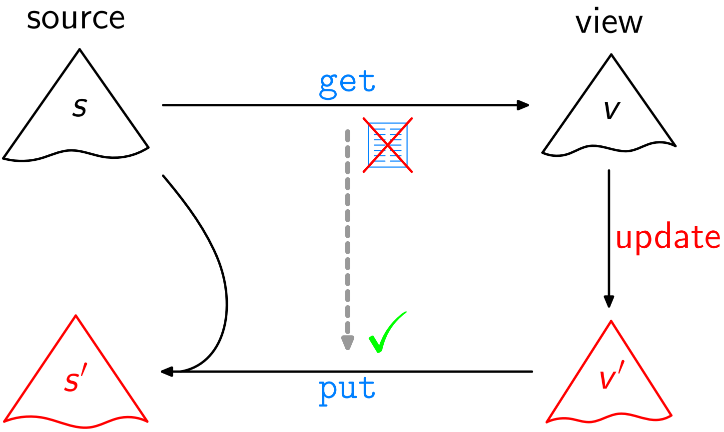
Bidirectional Transformation



Syntactic Bidirectionalization

[Matsuda et al., ICFP'07]

Bidirectional Transformation



Semantic Bidirectionalization

[V., POPL'09]

The Constant-Complement Approach [Bancilhon & Spyratos, ACM TODS'81]

Given

$$\texttt{get} :: S \to V$$

The Constant-Complement Approach [Bancilhon & Spyratos, ACM TODS'81]

Given

$$\texttt{get} :: S \to V$$

define a C and

$$\texttt{res} :: S \to C$$

The Constant-Complement Approach [Bancilhon & Spyratos, ACM TODS'81]

Given

$$\texttt{get} :: S \to V$$

define a C and

$$\texttt{res} :: S \to C$$

such that

$$\texttt{paired} = \lambda s \to (\texttt{get}\ s, \texttt{res}\ s)$$

is injective

The Constant-Complement Approach [Bancilhon & Spyratos, ACM TODS'81]

Given

$$\texttt{get} :: S \rightarrow V$$

define a C and

$$\texttt{res} :: S \rightarrow C$$

such that

$$\texttt{paired} = \lambda s \rightarrow (\texttt{get}\ s, \texttt{res}\ s)$$

is injective and has an inverse $\texttt{inv} :: (V, C) \rightarrow S$.

The Constant-Complement Approach [Bancilhon & Spyratos, ACM TODS'81]

Given

$$\texttt{get} :: S \to V$$

define a C and

$$\texttt{res} :: S \to C$$

such that

$$\texttt{paired} = \lambda s \to (\texttt{get}\ s, \texttt{res}\ s)$$

is injective and has an inverse $\texttt{inv} :: (V, C) \to S$.

Then:

$$\texttt{put} :: V \to S \to S$$
$$\texttt{put}\ v'\ s = \texttt{inv}\ (v', \texttt{res}\ s)$$

The Constant-Complement Approach

Guarantees "reasonability":

- $\texttt{put}\ (\texttt{get}\ s)\ s = s$
- $\texttt{get}\ (\texttt{put}\ v'\ s) = v'$
- $\texttt{put}\ v''\ (\texttt{put}\ v'\ s) = \texttt{put}\ v''\ s$

The Constant-Complement Approach

Guarantees "reasonability":

- $\texttt{put}\ (\texttt{get}\ s)\ s = s$
- $\texttt{get}\ (\texttt{put}\ v'\ s) = v'$
- $\texttt{put}\ v''\ (\texttt{put}\ v'\ s) = \texttt{put}\ v''\ s$

Example:

$$\texttt{get} :: \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2$$

The Constant-Complement Approach

Guarantees "reasonability":

- $\texttt{put}\ (\texttt{get}\ s)\ s = s$
- $\texttt{get}\ (\texttt{put}\ v'\ s) = v'$
- $\texttt{put}\ v''\ (\texttt{put}\ v'\ s) = \texttt{put}\ v''\ s$

Example:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

The Constant-Complement Approach

Guarantees "reasonability":

- $\texttt{put}\ (\texttt{get}\ s)\ s = s$
- $\texttt{get}\ (\texttt{put}\ v'\ s) = v'$
- $\texttt{put}\ v''\ (\texttt{put}\ v'\ s) = \texttt{put}\ v''\ s$

Example:

$$\begin{array}{ll} \texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} & \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2 \\ \texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 & \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2 \end{array}$$

$$\begin{array}{l} \texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}_2) \to \mathsf{Nat} \\ \texttt{inv}\ (v', c) = 2 * v' + c \end{array}$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \rightarrow \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \rightarrow \mathsf{Nat}_2$$

$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

$$\texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}_2) \rightarrow \mathsf{Nat}$$

$$\texttt{inv}\ (v', c) = 2 * v' + c$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \rightarrow \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \rightarrow \mathsf{Nat}_2$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

$$\texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}_2) \rightarrow \mathsf{Nat}$$
$$\texttt{inv}\ (v', c) = 2 * v' + c$$

Then:

$$\texttt{put} :: \mathsf{Nat} \rightarrow \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$\texttt{put}\ v'\ s = \texttt{inv}\ (v', \texttt{res}\ s)$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2$$

$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

$$\texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}_2) \to \mathsf{Nat}$$

$$\texttt{inv}\ (v', c) = 2 * v' + c$$

Then:

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$$

$$\begin{aligned}\texttt{put}\ v'\ s &= \texttt{inv}\ (v', \texttt{res}\ s) \\ &= 2 * v' + s\ \text{`}\texttt{mod}\text{`}\ 2\end{aligned}$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2$$

$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

Another choice for complement:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}$$

$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

Another choice for complement:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n$$

$$\texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}) \rightharpoonup \mathsf{Nat}$$
$$\texttt{inv}\ (v', c) \mid (v' == \texttt{get}\ c) = c$$

The Constant-Complement Approach

Example:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}_2$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n\ \text{`}\texttt{mod}\text{`}\ 2$$

Another choice for complement:

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat} \qquad \texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2 \qquad \texttt{res}\ n = n$$

$$\texttt{inv} :: (\mathsf{Nat}, \mathsf{Nat}) \rightharpoonup \mathsf{Nat}$$
$$\texttt{inv}\ (v', c) \mid (v' == \texttt{get}\ c) = c$$

Then:

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \rightharpoonup \mathsf{Nat}$$
$$\texttt{put}\ v'\ s \mid (v' == \texttt{get}\ s) = s$$

Catering for Partiality

Still require that `get` $:: S \to V$ and `res` $:: S \to C$ are total and that `paired` is injective.

Catering for Partiality

Still require that `get` $:: S \to V$ and `res` $:: S \to C$ are total and that `paired` is injective.

But allow `inv` $:: (V, C) \rightharpoonup S$, and instead of being a full inverse of `paired`, only require that:

- $\texttt{inv} \circ \texttt{paired} = \texttt{id}$
- $\texttt{paired} \circ \texttt{inv} \sqsubseteq \texttt{id}$

Catering for Partiality

Still require that get $:: S \to V$ and res $:: S \to C$ are total and that paired is injective.

But allow inv $:: (V, C) \rightharpoonup S$, and instead of being a full inverse of paired, only require that:

- $\texttt{inv} \circ \texttt{paired} = \texttt{id}$
- $\texttt{paired} \circ \texttt{inv} \sqsubseteq \texttt{id}$

Guarantees (only):

- $\texttt{put}\ (\texttt{get}\ s)\ s = s$
- $\texttt{get}\ (\texttt{put}\ v'\ s) \sqsubseteq v'$
- $(\texttt{put}\ v'\ s){\Downarrow} \Rightarrow \texttt{put}\ v''\ (\texttt{put}\ v'\ s) = \texttt{put}\ v''\ s$

Choices to Make

For

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{get}\ n = n \ \text{`}\texttt{div}\text{`}\ 2$$

clearly

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{put}\ v'\ s = 2 * v' + s \ \text{`}\texttt{mod}\text{`}\ 2$$

better than

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \rightharpoonup \mathsf{Nat}$$
$$\texttt{put}\ v'\ s \mid (v' == \texttt{get}\ s) = s$$

Choices to Make

For

$$\texttt{get} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{get}\ n = n\ \text{`}\texttt{div}\text{`}\ 2$$

clearly

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{put}\ v'\ s = 2 * v' + s\ \text{`}\texttt{mod}\text{`}\ 2$$

better than

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \rightharpoonup \mathsf{Nat}$$
$$\texttt{put}\ v'\ s \mid (v' == \texttt{get}\ s) = s$$

But what about:

$$\texttt{put} :: \mathsf{Nat} \to \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{put}\ v'\ s = 2 * v' + (v' + ((s + 1)\ \text{`}\texttt{mod}\text{`}\ 4)\ \text{`}\texttt{div}\text{`}\ 2)\ \text{`}\texttt{mod}\text{`}\ 2$$

Choices to Make

Different complement functions (`res`) lead to different update functions (`put`):

$v' \setminus s$	0	1	2	3
0	0	1	0	1
1	2	3	2	3
2	4	5	4	5
3	6	7	6	7

vs.

$v' \setminus s$	0	1	2	3
0	0	1	1	0
1	3	2	2	3
2	4	5	5	4
3	7	6	6	7

Choices to Make

Different complement functions (`res`) lead to different update functions (`put`):

$v' \setminus s$	0	1	2	3
0	0	1	0	1
1	2	3	2	3
2	4	5	4	5
3	6	7	6	7

vs.

$v' \setminus s$	0	1	2	3
0	0	1	1	0
1	3	2	2	3
2	4	5	5	4
3	7	6	6	7

In fact, `res` $:: S \rightarrow C$ is the only "handle" we have for influencing the choice of `put`.

Small Complements

The bad thing about

$$\texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{res}\ n = n$$

is that it is "too injective".

Small Complements

The bad thing about

$$\texttt{res} :: \mathsf{Nat} \rightarrow \mathsf{Nat}$$
$$\texttt{res}\ n = n$$

is that it is "too injective".

Note that we need

$$\texttt{paired} = \lambda s \rightarrow (\texttt{get}\ s, \texttt{res}\ s)$$

to be injective, but not `res` itself.

Small Complements

The bad thing about

$$\texttt{res} :: \mathsf{Nat} \to \mathsf{Nat}$$
$$\texttt{res}\ n = n$$

is that it is "too injective".

Note that we need

$$\texttt{paired} = \lambda s \to (\texttt{get}\ s, \texttt{res}\ s)$$

to be injective, but not res itself.

In fact, the "less injective", the better!

Small Complements

Formally:

$$\texttt{res}_1 \preceq \texttt{res}_2 \quad \Leftrightarrow \quad (\mathsf{ker}\ \texttt{res}_2) \subseteq (\mathsf{ker}\ \texttt{res}_1)$$

Small Complements

Formally:

$$\texttt{res}_1 \preceq \texttt{res}_2 \quad \Leftrightarrow \quad (\text{ker } \texttt{res}_2) \subseteq (\text{ker } \texttt{res}_1)$$

Clearly fulfilled for:

$$\texttt{res}_1 :: \mathsf{Nat} \to \mathsf{Nat}_2 \qquad\qquad \texttt{res}_2 :: \mathsf{Nat} \to \mathsf{Nat}$$

$$\texttt{res}_1\; n = n \;\text{`}\texttt{mod}\text{`}\; 2 \qquad\qquad \texttt{res}_2\; n = n$$

Small Complements

Formally:

$$\texttt{res}_1 \preceq \texttt{res}_2 \quad \Leftrightarrow \quad (\text{ker } \texttt{res}_2) \subseteq (\text{ker } \texttt{res}_1)$$

Clearly fulfilled for:

$$\texttt{res}_1 :: \text{Nat} \to \text{Nat}_2 \qquad \texttt{res}_2 :: \text{Nat} \to \text{Nat}$$

$$\texttt{res}_1\ n = n \text{ `}\texttt{mod}\text{` } 2 \qquad \texttt{res}_2\ n = n$$

Theorem [Bancilhon & Spyratos, ACM TODS'81]:
For given $\texttt{get} :: S \to V$,

$$\texttt{res}_1 \preceq \texttt{res}_2 \quad \Leftrightarrow \quad \forall v', s.\ \texttt{put}_2\ v'\ s \sqsubseteq \texttt{put}_1\ v'\ s$$

Summary of the Approach to Bidirectionalization

Given `get` $:: S \rightarrow V$, find C and `res` $:: S \rightarrow C$ such that `paired` $= \lambda s \rightarrow ($`get` $s,$ `res` $s)$ is injective and `res` is as small as possible with respect to $\preceq$.

Summary of the Approach to Bidirectionalization

Given `get` $:: S \to V$, find C and `res` $:: S \to C$ such that `paired` $= \lambda s \to ($`get` $s,$ `res` $s)$ is injective and `res` is as small as possible with respect to $\preceq$.

Define (an effective!) `inv` $:: (V, C) \rightharpoonup S$ with:

$$\texttt{inv}\ (v', c) = \begin{cases} \bot & \text{if } \neg\exists s'.\ \texttt{paired}\ s' = (v', c) \\ s' & \text{if } \texttt{paired}\ s' = (v', c) \end{cases}$$

Summary of the Approach to Bidirectionalization

Given $\texttt{get} :: S \to V$, find C and $\texttt{res} :: S \to C$ such that $\texttt{paired} = \lambda s \to (\texttt{get}\ s, \texttt{res}\ s)$ is injective and $\texttt{res}$ is as small as possible with respect to $\preceq$.

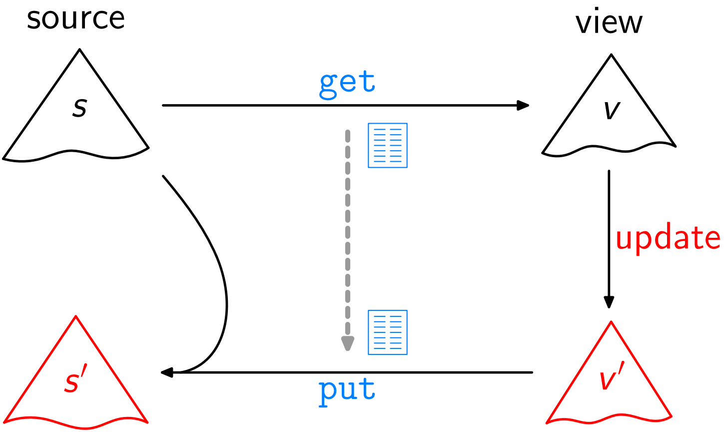
Define (an effective!) $\texttt{inv} :: (V, C) \rightharpoonup S$ with:

$$\texttt{inv}\ (v', c) = \begin{cases} \bot & \text{if } \neg \exists s'.\ \texttt{paired}\ s' = (v', c) \\ s' & \text{if } \texttt{paired}\ s' = (v', c) \end{cases}$$

Set:

$$\begin{aligned} &\texttt{put} :: V \to S \rightharpoonup S \\ &\texttt{put}\ v'\ s = \texttt{inv}\ (v', \texttt{res}\ s) \end{aligned}$$

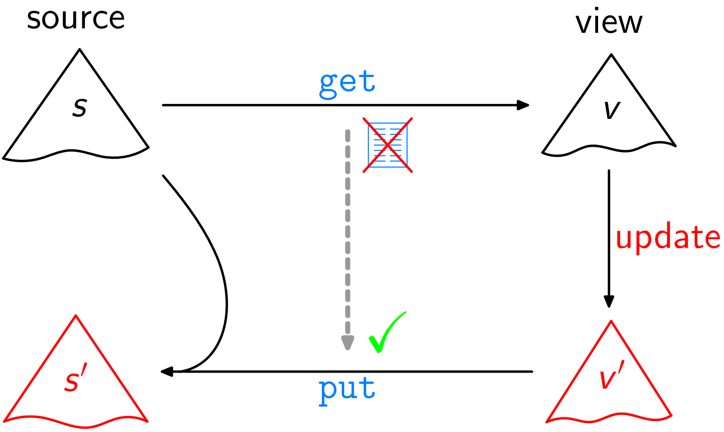
Bidirectional Transformation



Syntactic Bidirectionalization

[Matsuda et al., ICFP'07]

Bidirectional Transformation



Semantic Bidirectionalization

[V., POPL'09]

Taking Stock

[Matsuda et al., ICFP'07]:

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

[V., POPL'09]:

- very lightweight, easy access to bidirectionality
- essential role: polymorphic function types
- major problem: rejects shape-changing updates

[V. et al., ICFP'10]:

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

Scorecard

	syntactic	semantic	combined
Update?	State-based		
Bijective?	No		
Well behaved?	Yes		
Very well behaved?	Yes		No
Choice of `put`?	No		Yes
Total?	No		

References I

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

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.

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.

References II

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.