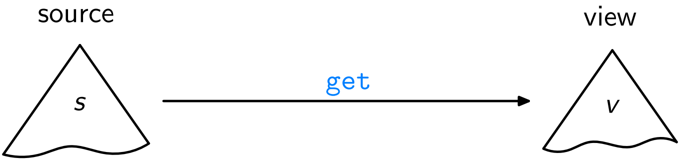
Inductive Program Synthesis for Bidirectional Transformations

Tobias Gödderz, Helmut Grohne, Janis Voigtländer

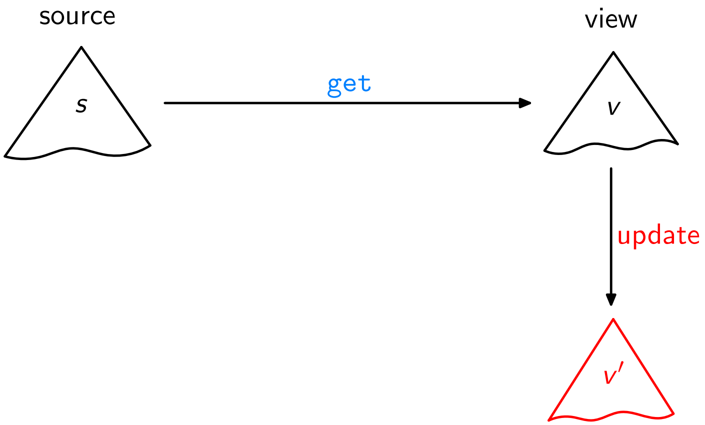
University of Bonn

Dagstuhl Seminar 15442

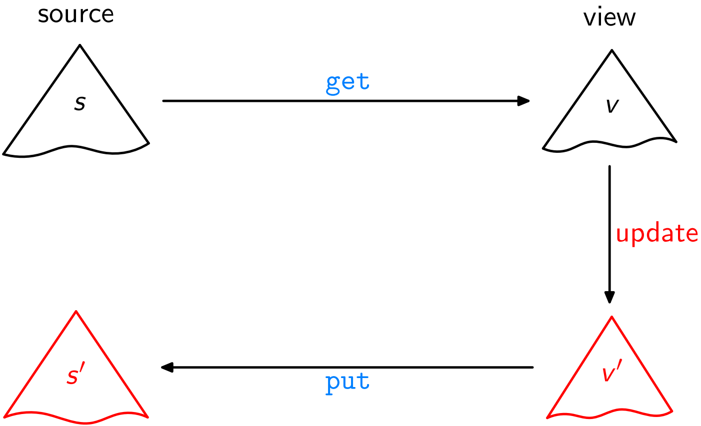
Bidirectional Transformation



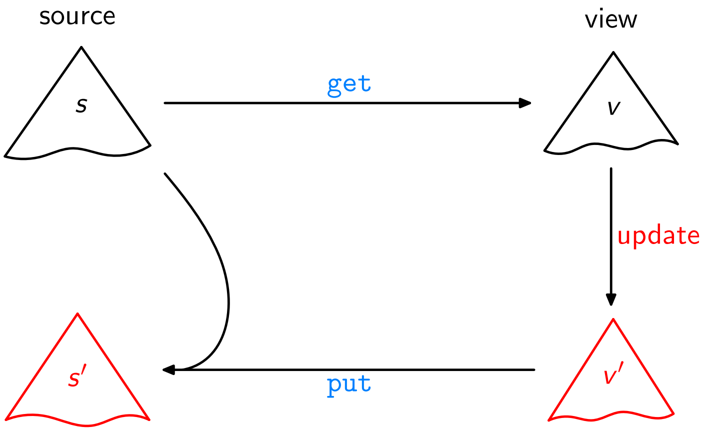
Bidirectional Transformation



Bidirectional Transformation



Bidirectional Transformation



Bidirectional Transformation

Examples:

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

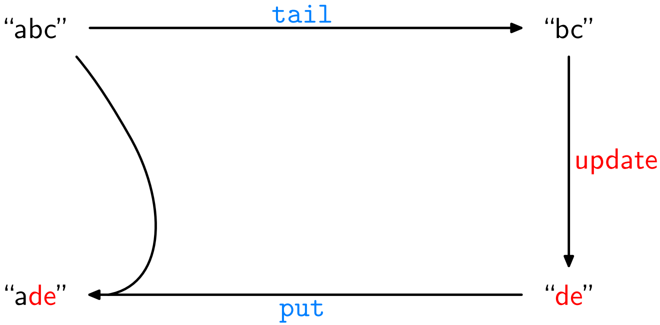
Bidirectional Transformation

Examples:

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

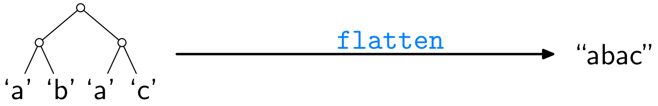
Bidirectional Transformation

Examples:



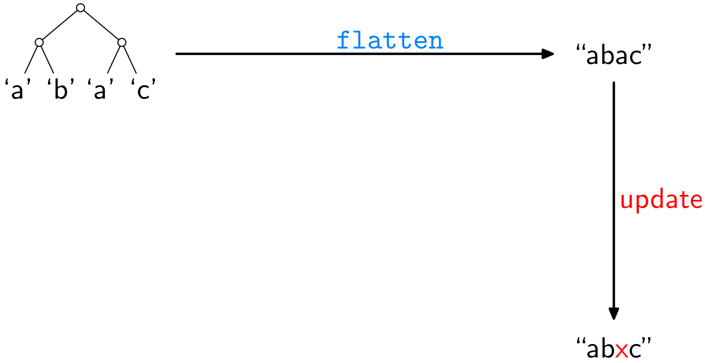
Bidirectional Transformation

Examples:



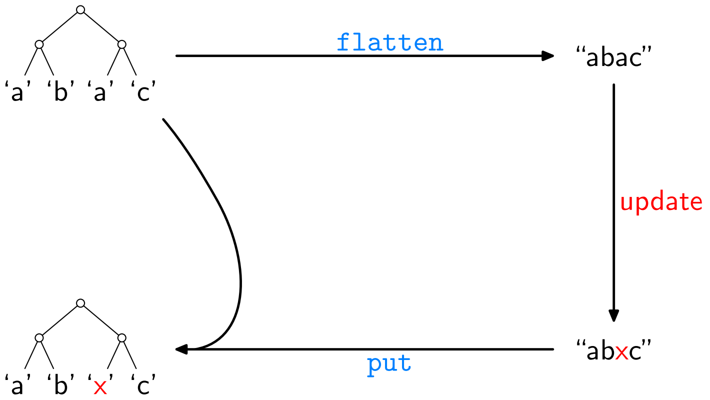
Bidirectional Transformation

Examples:



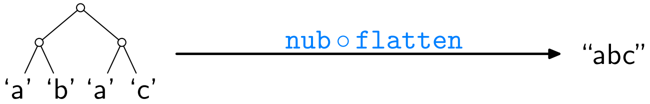
Bidirectional Transformation

Examples:



Bidirectional Transformation

Examples:

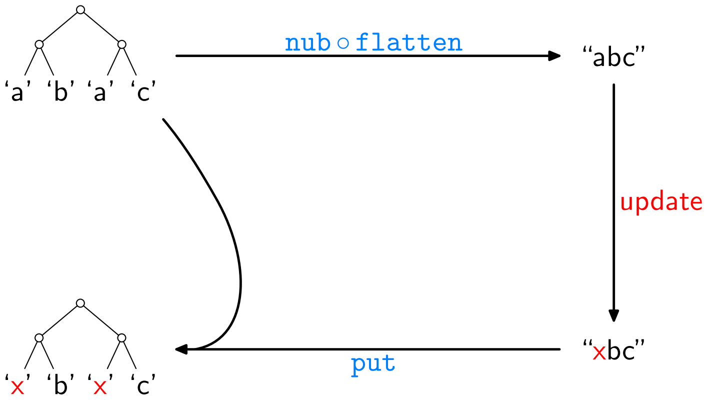


Bidirectional Transformation

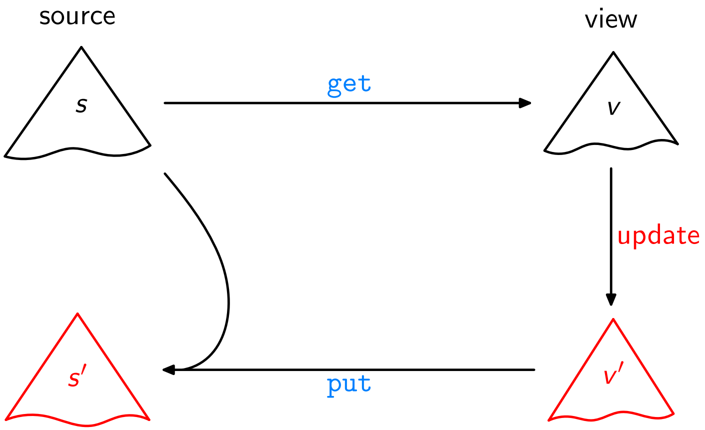
Examples:

Bidirectional Transformation

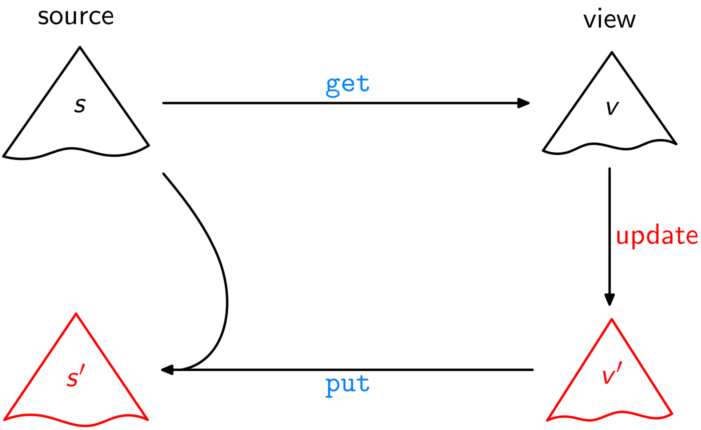
Examples:



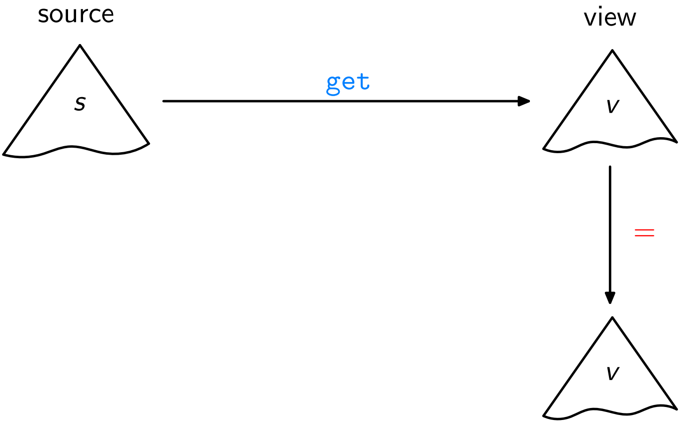
Bidirectional Transformation



Bidirectional Transformation – Laws

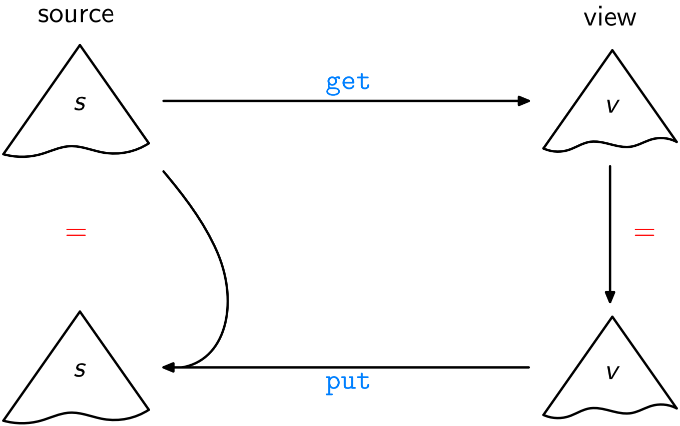


Bidirectional Transformation – Laws



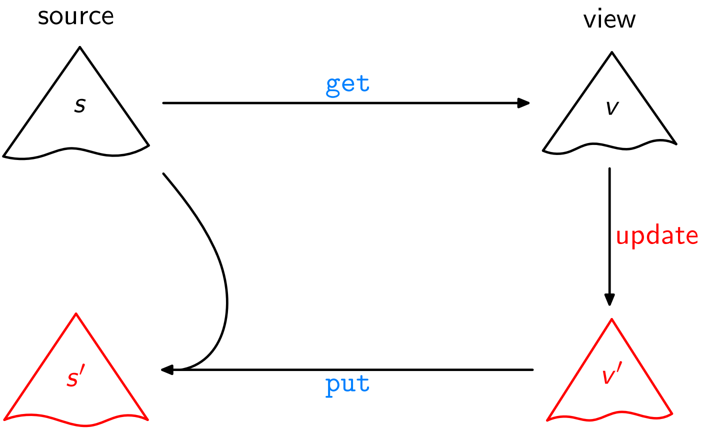
Acceptability / GetPut

Bidirectional Transformation – Laws



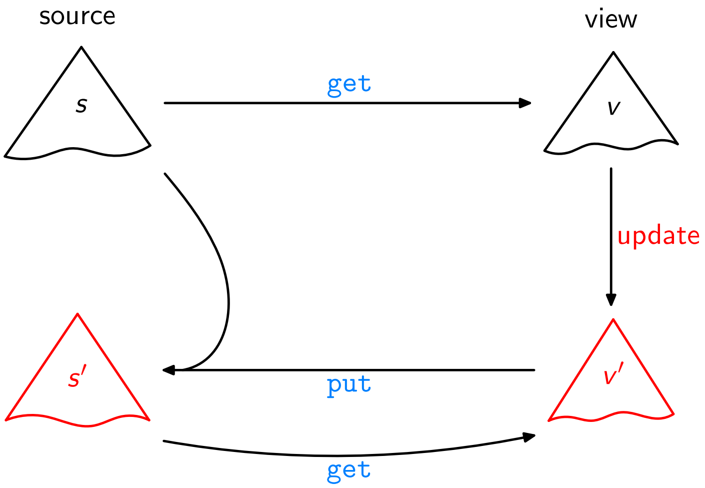
Acceptability / GetPut

Bidirectional Transformation – Laws



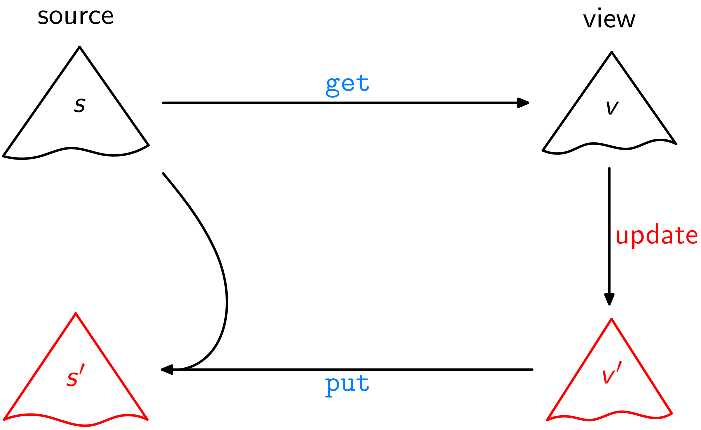
Consistency / PutGet

Bidirectional Transformation – Laws

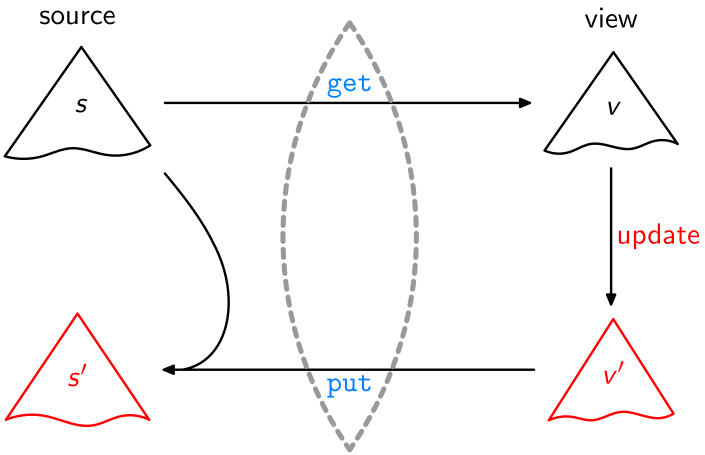


Consistency / PutGet

Bidirectional Transformation



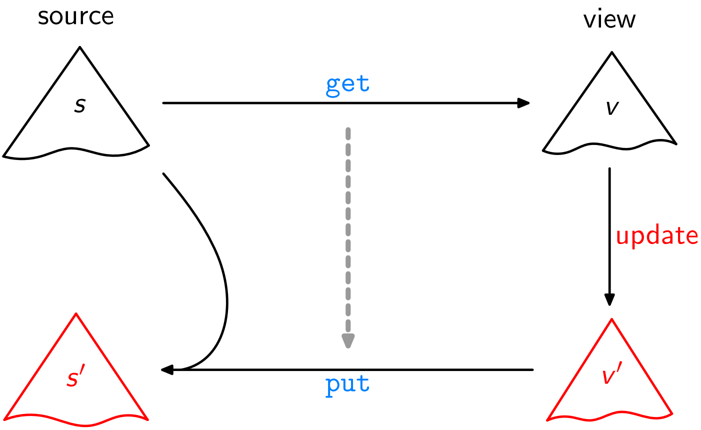
Bidirectional Transformation – PL Approaches



Lenses, DSLs

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

Bidirectional Transformation – PL Approaches



Bidirectionalization

[Matsuda et al., ICFP'07], [V., POPL'09], ...

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	""	"a"	"ab"	"abc"	"abcd"	"abcde"
`sieve` s	""	""	"b"	"b"	"bd"	"bd"

Nondeterminism / Choices to make

Let `get` $=$ `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

`put "abcd" "xy"` $=$ `"axcy"`

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	""	"a"	"ab"	"abc"	"abcd"	"abcde"
`sieve` s	""	""	"b"	"b"	"bd"	"bd"

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" =
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	""	"a"	"ab"	"abc"	"abcd"	"abcde"
`sieve` s	""	""	"b"	"b"	"bd"	"bd"

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez  "?
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez "?
put "abcd"  "xyz" = "axcy z"
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez "?
put "abcd"  "xyz" = "axcy z"
put "abcd"  "x"   =
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	""	"a"	"ab"	"abc"	"abcd"	"abcde"
`sieve` s	""	""	"b"	"b"	"bd"	"bd"

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez  "?
put "abcd"  "xyz" = "axcy z"
put "abcd"  "x"   = "axc"
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez  "?
put "abcd"  "xyz" = "axcy z"
put "abcd"  "x"   = "axc"        or "ax"?
```

Nondeterminism / Choices to make

Let `get` = `sieve` with:

s	`""`	`"a"`	`"ab"`	`"abc"`	`"abcd"`	`"abcde"`
`sieve` s	`""`	`""`	`"b"`	`"b"`	`"bd"`	`"bd"`

Then, for example:

```
put "abcd"  "xy"  = "axcy"
put "abcde" "xy"  = "axcye"
put "abcde" "xyz" = "axcyez"          or "axcyez  "?
put "abcd"  "xyz" = "axcy z"
put "abcd"  "x"   = "axc"        or "ax"?,   or "cx"?
```

Nondeterminism / Choices to make

Let `get` $=$ `head` with:

$$\texttt{head}\ (x : xs) = x$$

Nondeterminism / Choices to make

Let `get` = `head` with:

$$\texttt{head}\ (x : xs) = x$$

Maybe:

$$\texttt{put}\ (x : xs)\ y = [y]$$

Nondeterminism / Choices to make

Let $\texttt{get} = \texttt{head}$ with:

$$\texttt{head}\ (x : xs) = x$$

Maybe:

$$\texttt{put}\ (x : xs)\ y = [y]$$

But that violates $\texttt{put}\ s\ (\texttt{get}\ s) = s\,!$

Nondeterminism / Choices to make

Let $\texttt{get} = \texttt{head}$ with:

$$\texttt{head}\ (x : xs) = x$$

Maybe:

$$\texttt{put}\ (x : xs)\ y = [y]$$

But that violates $\texttt{put}\ s\ (\texttt{get}\ s) = s$!

Better:

$$\begin{aligned} \texttt{put}\ (x : xs)\ y \mid y == x &= (x : xs) \\ \mid \texttt{otherwise} &= [y] \end{aligned}$$

Nondeterminism / Choices to make

Let $\texttt{get} = \texttt{head}$ with:

$$\texttt{head}\ (x : xs) = x$$

Maybe:

$$\texttt{put}\ (x : xs)\ y = [y]$$

But that violates $\texttt{put}\ s\ (\texttt{get}\ s) = s$!

Better:

$$\begin{aligned} \texttt{put}\ (x : xs)\ y\ &|\ y == x \quad\ = (x : xs) \\ &|\ \texttt{otherwise} = [y] \end{aligned}$$

But "really intended":

$$\texttt{put}\ (x : xs)\ y = (y : xs)$$

A slightly more complex case, with recursion

Let $\texttt{get} = \texttt{init}$ with:

$$\begin{aligned} \texttt{init}\,[x] \quad &= [\,] \\ \texttt{init}\,(x : xs) &= (x : (\texttt{init}\,xs)) \end{aligned}$$

A slightly more complex case, with recursion

Let $\texttt{get} = \texttt{init}$ with:

$$\begin{aligned} \texttt{init}\,[x] &= [\,] \\ \texttt{init}\,(x : xs) &= (x : (\texttt{init}\; xs)) \end{aligned}$$

Possible, and correct:

$$\begin{aligned} \texttt{put}\; xs\; ys \mid \texttt{length}\; ys == (\texttt{length}\; xs) - 1 &= ys +\!\!+ [\texttt{last}\; xs] \\ \mid \texttt{otherwise} &= ys +\!\!+ \texttt{" "} \end{aligned}$$

A slightly more complex case, with recursion

Let $\texttt{get} = \texttt{init}$ with:

$$\begin{aligned} \texttt{init}\ [x] \quad &= [\,] \\ \texttt{init}\ (x : xs) &= (x : (\texttt{init}\ xs)) \end{aligned}$$

Possible, and correct:

$$\begin{aligned} \texttt{put}\ xs\ ys \mid \texttt{length}\ ys == (\texttt{length}\ xs) - 1 &= ys +\!\!+ [\texttt{last}\ xs] \\ \mid \texttt{otherwise} \qquad\qquad\qquad &= ys +\!\!+ \texttt{" "} \end{aligned}$$

But intended:

$$\texttt{put}\ xs\ ys = ys +\!\!+ [\texttt{last}\ xs]$$

A slightly more complex case, with recursion

Let $\texttt{get} = \texttt{init}$ with:

$$\begin{aligned}\texttt{init}\,[x] \quad &= [\,]\\ \texttt{init}\,(x : xs) &= (x : (\texttt{init}\,xs))\end{aligned}$$

Possible, and correct:

$$\begin{aligned}\texttt{put}\ xs\ ys \mid\ &\texttt{length}\ ys == (\texttt{length}\ xs) - 1 &&= ys +\!\!+ [\texttt{last}\ xs]\\ \mid\ &\texttt{otherwise} &&= ys +\!\!+ \texttt{"\ "}\end{aligned}$$

But intended:

$$\texttt{put}\ xs\ ys = ys +\!\!+ [\texttt{last}\ xs]$$

Problem: How to produce the "intuitive" solution?

Entry: Inductive Program Synthesis

Entry: Inductive Program Synthesis

Problem: Of the well-behavedness laws

$$\texttt{put}\ s\ (\texttt{get}\ s) = s$$

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

only the first one directly delivers I/O pairs for `put`.

Entry: Inductive Program Synthesis

Problem: Of the well-behavedness laws

$$\texttt{put}\ s\ (\texttt{get}\ s) = s$$

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

only the first one directly delivers I/O pairs for `put`.

Like, for $\texttt{get} = \texttt{init}$:

$$\begin{aligned}
&\texttt{put}\ [a] && [\,] && = [a]\\
&\texttt{put}\ [a,b] && [a] && = [a,b]\\
&\texttt{put}\ [a,b,c] && [a,b] && = [a,b,c]\\
&\texttt{put}\ [a,b,c,d] && [a,b,c] && = [a,b,c,d]
\end{aligned}$$

Entry: Inductive Program Synthesis

Problem: Of the well-behavedness laws

$$\texttt{put}\ s\ (\texttt{get}\ s) = s$$

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

only the first one directly delivers I/O pairs for `put`.

Like, for $\texttt{get} = \texttt{init}$:

$$\begin{array}{llll} \texttt{put} & [a] & [\,] & = [a] \\ \texttt{put} & [a,b] & [a] & = [a,b] \\ \texttt{put} & [a,b,c] & [a,b] & = [a,b,c] \\ \texttt{put} & [a,b,c,d] & [a,b,c] & = [a,b,c,d] \end{array}$$

But then one would synthesize:

$$\texttt{put}\ xs\ ys = xs$$

One possible approach

"Encourage" use of both arguments!

One possible approach

"Encourage" use of both arguments!

Instead of:

$$\begin{array}{llll} \texttt{put}\ [a] & [\,] & = [a] \\ \texttt{put}\ [a, b] & [a] & = [a, b] \\ \ldots & & \end{array}$$

use:

$$\begin{array}{llll} \texttt{put}\ [a] & [\,] & = [a] \\ \texttt{put}\ [\_, b] & [a] & = [a, b] \\ \texttt{put}\ [\_, \_, c] & [a, b] & = [a, b, c] \\ \texttt{put}\ [\_, \_, \_, d] & [a, b, c] & = [a, b, c, d] \end{array}$$

One possible approach

"Encourage" use of both arguments!

Instead of:

$$\begin{aligned}
&\texttt{put}\ [a] && [\,] && = [a]\\
&\texttt{put}\ [a, b] && [a] && = [a, b]\\
&\ldots
\end{aligned}$$

use:

$$\begin{aligned}
&\texttt{put}\ [a] && [\,] && = [a]\\
&\texttt{put}\ [_, b] && [a] && = [a, b]\\
&\texttt{put}\ [_, _, c] && [a, b] && = [a, b, c]\\
&\texttt{put}\ [_, _, _, d] && [a, b, c] && = [a, b, c, d]
\end{aligned}$$

Then, Igor II synthesizes:

$$\begin{aligned}
&\texttt{put}\ [a] && [\,] && = [a]\\
&\texttt{put}\ (_ : (x : ys)) && (a : bs) && = (a : (\texttt{put}\ (x : ys)\ bs))
\end{aligned}$$

One possible approach

Also works for $\texttt{get} = \texttt{sieve}$. Gives:

$$
\begin{aligned}
&\texttt{put}\ xs && [\,] && = xs \\
&\texttt{put}\ (a : (\_ : xs)) && (b : ys) && = (a : (b : (\texttt{put}\ xs\ ys)))
\end{aligned}
$$

One possible approach

Also works for $\texttt{get} = \texttt{sieve}$. Gives:

$$
\begin{array}{lll}
\texttt{put}\ \mathit{xs} & [\,] & = \mathit{xs} \\
\texttt{put}\ (a : (\_ : \mathit{xs}))\ & (b : \mathit{ys}) & = (a : (b : (\texttt{put}\ \mathit{xs}\ \mathit{ys})))
\end{array}
$$

But this $\texttt{put}$ is not defined when $(\texttt{length}\ s)\ /\ 2 < \texttt{length}\ v'$.

One possible approach

Also works for $\texttt{get} = \texttt{sieve}$. Gives:

$$\begin{aligned}
&\texttt{put}\ xs && [\,] && = xs \\
&\texttt{put}\ (a : (_ : xs)) && (b : ys) && = (a : (b : (\texttt{put}\ xs\ ys)))
\end{aligned}$$

But this $\texttt{put}$ is not defined when $(\texttt{length}\ s)\ /\ 2 < \texttt{length}\ v'$.

Idea: Introduce extra examples covering such cases:

$$\texttt{put}\ [\,]\ [b] = [\_, b]$$

(as a “mutation” of $\texttt{put}\ [a, \_]\ [b] = [a, b]$).

One possible approach

Also works for $\texttt{get} = \texttt{sieve}$. Gives:

$$
\begin{aligned}
&\texttt{put}\ \mathit{xs} && [\,] && = \mathit{xs}\\
&\texttt{put}\ (a : (\_ : \mathit{xs})) && (b : \mathit{ys}) && = (a : (b : (\texttt{put}\ \mathit{xs}\ \mathit{ys})))
\end{aligned}
$$

But this $\texttt{put}$ is not defined when $(\texttt{length}\ s) \,/\, 2 < \texttt{length}\ v'$.

Idea: Introduce extra examples covering such cases:

$$\texttt{put}\ [\,]\ [b] = [\_, b]$$

(as a "mutation" of $\texttt{put}\ [a, \_]\ [b] = [a, b]$).

But actually then, in general, also need to express inequality constraints . . .

Conclusion / Outlook

- Bidirectional Transformations:
  - “hot topic” in various areas, including PL approaches
  - typical weakness: nondeterminism, and limited (or no) impact of programmer intentions

Conclusion / Outlook

- Bidirectional Transformations:
  - "hot topic" in various areas, including PL approaches
  - typical weakness: nondeterminism, and limited (or no) impact of programmer intentions
- Connection to Inductive Programming:
  - IP as a "helper", detecting/exploiting regularities
  - either naively as a black box, or deeper integration
  - further ideas: I/O pairs per parametricity of `get`;
    user impact through ad-hoc I/O pairs or provision of background knowledge;
    ...

Conclusion / Outlook

- Bidirectional Transformations:
  - "hot topic" in various areas, including PL approaches
  - typical weakness: nondeterminism, and limited (or no) impact of programmer intentions
- Connection to Inductive Programming:
  - IP as a "helper", detecting/exploiting regularities
  - either naively as a black box, or deeper integration
  - further ideas: I/O pairs per parametricity of `get`;
    user impact through ad-hoc I/O pairs or provision of background knowledge;
    ...
- Extensions to Igor II:
  - dealing with wildcards on rhs of I/O pairs
  - a new operator for introducing accumulating parameters
  - some reduction of search space

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.

S. Katayama.
Systematic search for lambda expressions.
In *Trends in Functional Programming 2005, Revised Selected Papers*, pages 111–126. Intellect, 2007.

References II

E. Kitzelmann and U. Schmid.
Inductive synthesis of functional programs: An explanation based generalization approach.
Journal of Machine Learning Research, 7:429–454, 2006.

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