Ideas for Connecting Inductive Program Synthesis and Bidirectionalization

Janis Voigtländer

University of Bonn

PEPM'12

A small "test"

Which function is this?

$$
\begin{aligned}
\texttt{f}_1\ [a] &= a \\
\texttt{f}_1\ [a, b] &= b \\
\texttt{f}_1\ [a, b, c] &= c \\
\texttt{f}_1\ [a, b, c, d] &= d
\end{aligned}
$$

A small "test"

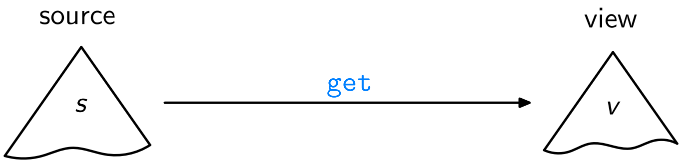
Which function is this?

$$\begin{aligned} \mathtt{f}_1\ [a] &= a \\ \mathtt{f}_1\ [a, b] &= b \\ \mathtt{f}_1\ [a, b, c] &= c \\ \mathtt{f}_1\ [a, b, c, d] &= d \end{aligned}$$

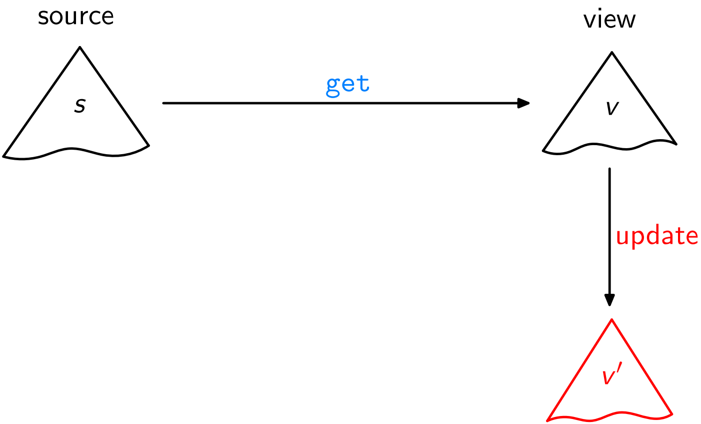
And this one?

$$\begin{aligned} \mathtt{f}_2\ [\,] &= [\,] \\ \mathtt{f}_2\ [a] &= [a] \\ \mathtt{f}_2\ [a, b] &= [b, a] \\ \mathtt{f}_2\ [a, b, c] &= [c, b, a] \end{aligned}$$

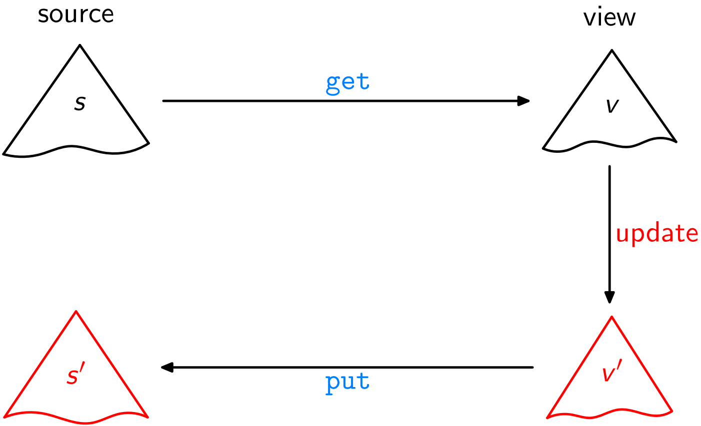
View-Update [Banc. & Sp., ACM TODS'81]



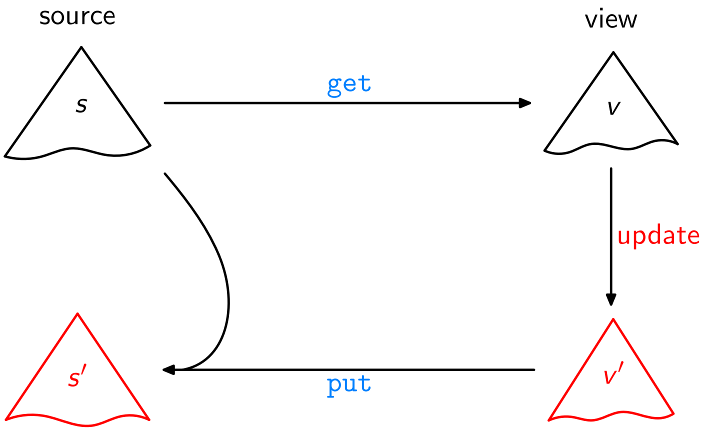
View-Update [Banc. & Sp., ACM TODS'81]



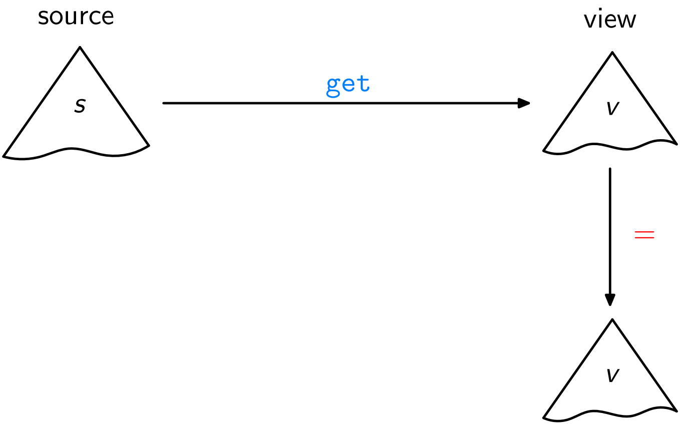
View-Update [Banc. & Sp., ACM TODS'81]



View-Update [Banc. & Sp., ACM TODS'81]

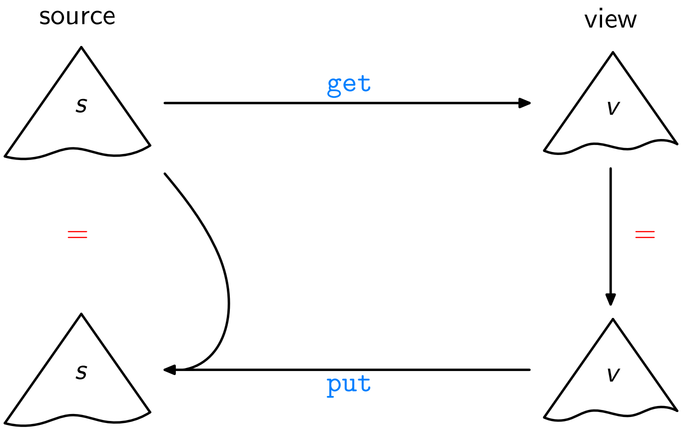


View-Update [Banc. & Sp., ACM TODS'81]



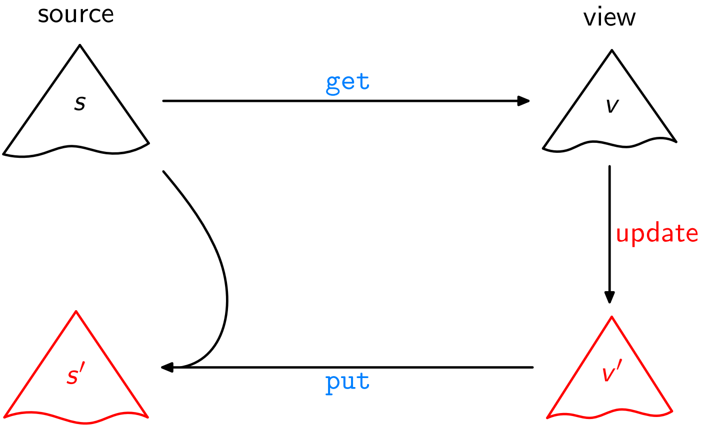
Acceptability / GetPut

View-Update [Banc. & Sp., ACM TODS'81]



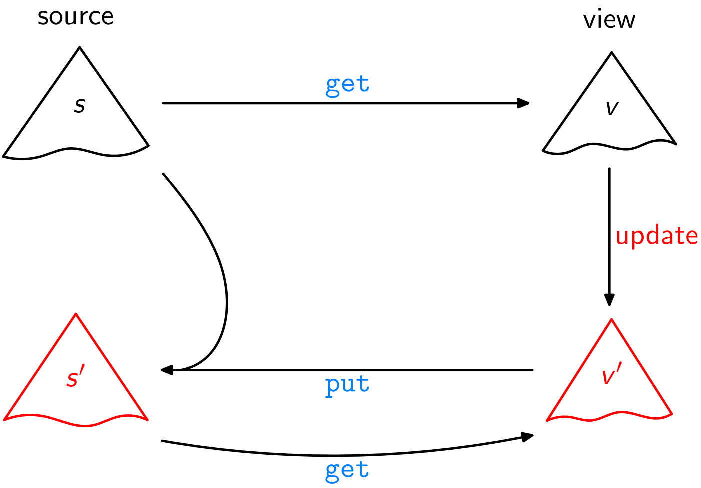
Acceptability / GetPut

View-Update [Banc. & Sp., ACM TODS'81]



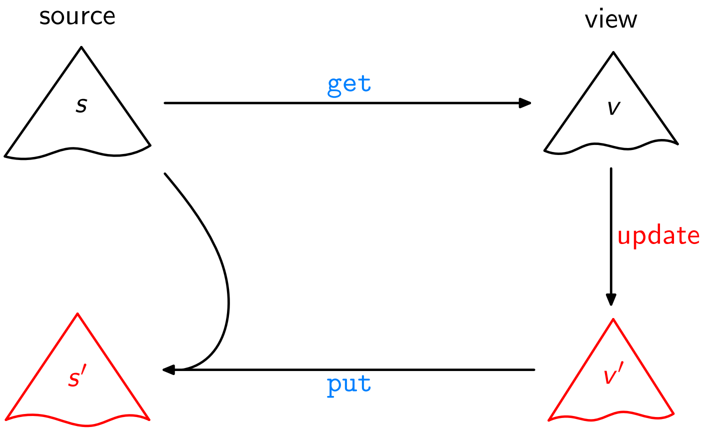
Consistency / PutGet

View-Update [Banc. & Sp., ACM TODS'81]

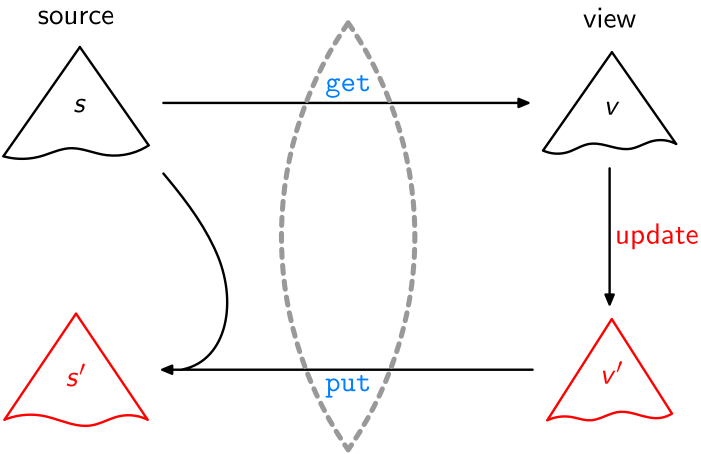


Consistency / PutGet

View-Update [Banc. & Sp., ACM TODS'81]



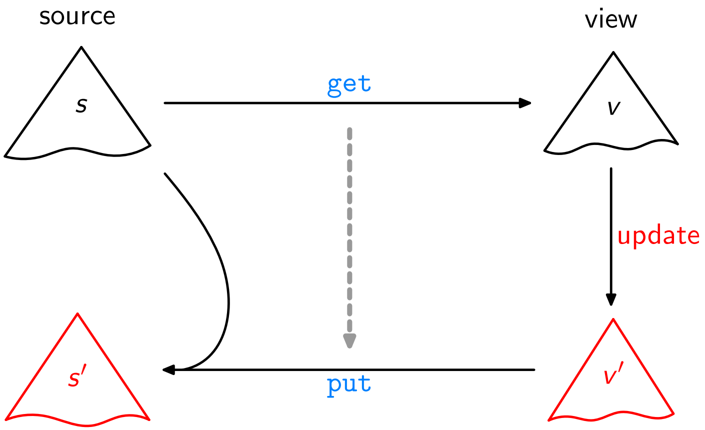
View-Update [Banc. & Sp., ACM TODS'81]



Lenses, DSLs

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

View-Update [Banc. & Sp., ACM TODS'81]



Bidirectionalization

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

Bidirectionalization (BX)

Examples:

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

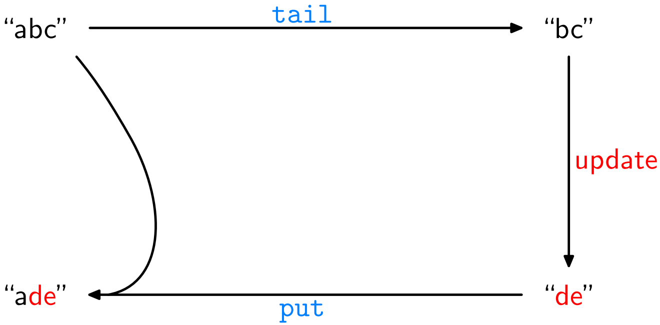
Bidirectionalization (BX)

Examples:

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

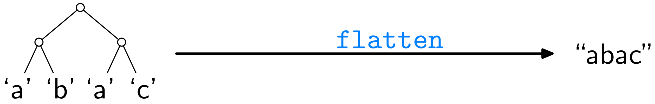
Bidirectionalization (BX)

Examples:



Bidirectionalization (BX)

Examples:

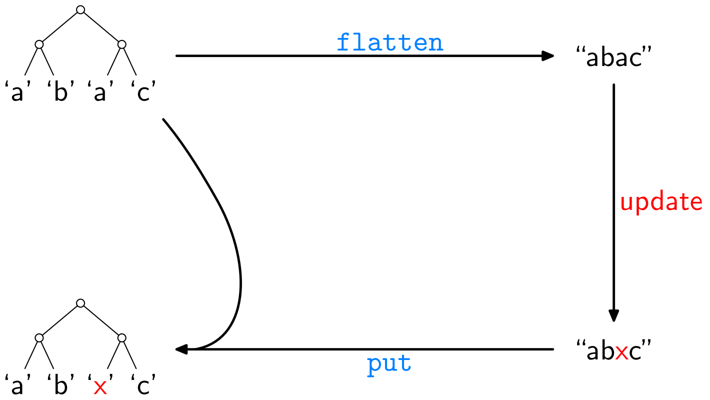


Bidirectionalization (BX)

Examples:

Bidirectionalization (BX)

Examples:



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:

$$\texttt{put}\ \texttt{"abcd"}\ \ \texttt{"xy"} = \texttt{"axcy"}$$

$$\texttt{put}\ \texttt{"abcde"}\ \texttt{"xy"} = \texttt{"axcye"}$$

$$\texttt{put}\ \texttt{"abcde"}\ \texttt{"xyz"} = \texttt{"axcyez"}$$

Nondeterminism / Choices to make

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

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

Then, for example:

$\texttt{put}\ \texttt{"abcd"}\ \ \texttt{"xy"} = \texttt{"axcy"}$

$\texttt{put}\ \texttt{"abcde"}\ \texttt{"xy"} = \texttt{"axcye"}$

$\texttt{put}\ \texttt{"abcde"}\ \texttt{"xyz"} = \texttt{"axcyez"}$ 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 $\texttt{get} = \texttt{sieve}$ 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 $\texttt{get} = \texttt{head}$ 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 $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs\,!$

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 $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs\,!$

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 $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs\,!$

Better:

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

But "really intended":

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

A slightly more complex case, with recursion

Let `get` $=$ `init` with:

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

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:

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

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:

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

But intended:

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

A slightly more complex case, with recursion

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

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

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 guide/control the possible choices?

Entry: Inductive Program Synthesis (IP)

Recall, I/O pairs for a function:

$$
\begin{aligned}
\texttt{f}_1\ [a] &= a \\
\texttt{f}_1\ [a, b] &= b \\
\texttt{f}_1\ [a, b, c] &= c \\
\texttt{f}_1\ [a, b, c, d] &= d
\end{aligned}
$$

Entry: Inductive Program Synthesis (IP)

Recall, I/O pairs for a function:

$$
\begin{aligned}
\mathtt{f_1}\ [a] &= a \\
\mathtt{f_1}\ [a, b] &= b \\
\mathtt{f_1}\ [a, b, c] &= c \\
\mathtt{f_1}\ [a, b, c, d] &= d
\end{aligned}
$$

From this, an IP system automatically generates the program:

$$
\begin{aligned}
\mathtt{f_1}\ [x] &= x \\
\mathtt{f_1}\ (x : xs) &= \mathtt{f_1}\ xs
\end{aligned}
$$

Entry: Inductive Program Synthesis (IP)

Recall, I/O pairs for a function:

$$
\begin{aligned}
\mathtt{f}_1\ [a] &= a \\
\mathtt{f}_1\ [a, b] &= b \\
\mathtt{f}_1\ [a, b, c] &= c \\
\mathtt{f}_1\ [a, b, c, d] &= d
\end{aligned}
$$

From this, an IP system automatically generates the program:

$$
\begin{aligned}
\mathtt{f}_1\ [x] &= x \\
\mathtt{f}_1\ (x : xs) &= \mathtt{f}_1\ xs
\end{aligned}
$$

Or:

$$
\begin{aligned}
\mathtt{f}_2\ [\,] &= [\,] \\
\mathtt{f}_2\ [a] &= [a] \\
\mathtt{f}_2\ [a, b] &= [b, a] \\
\mathtt{f}_2\ [a, b, c] &= [c, b, a]
\end{aligned}
$$

Entry: Inductive Program Synthesis (IP)

Or:

$$
\begin{aligned}
\mathtt{f}_2\,[\,] &= [\,] \\
\mathtt{f}_2\,[a] &= [a] \\
\mathtt{f}_2\,[a, b] &= [b, a] \\
\mathtt{f}_2\,[a, b, c] &= [c, b, a]
\end{aligned}
$$

Again automatically generated:

$$
\begin{aligned}
\mathtt{f}_2\,[\,] &= [\,] \\
\mathtt{f}_2\,(x : xs) &= ((\mathtt{f}_3\,(x : xs)) : (\mathtt{f}_2\,(\mathtt{f}_4\,(x : xs)))) \\
\mathtt{f}_3\,[x] &= x \\
\mathtt{f}_3\,(x : xs) &= \mathtt{f}_3\,xs \\
\mathtt{f}_4\,[x] &= [\,] \\
\mathtt{f}_4\,(x : xs) &= (x : (\mathtt{f}_4\,xs))
\end{aligned}
$$

Entry: Inductive Program Synthesis (IP)

Or:

$$
\begin{aligned}
\texttt{f}_2\,[\,] &= [\,] \\
\texttt{f}_2\,[a] &= [a] \\
\texttt{f}_2\,[a, b] &= [b, a] \\
\texttt{f}_2\,[a, b, c] &= [c, b, a]
\end{aligned}
$$

Or, through provision of `snoc` as "background knowledge":

$$
\begin{aligned}
\texttt{f}_2\,[\,] &= [\,] \\
\texttt{f}_2\,(x : xs) &= \texttt{snoc}\,(\texttt{f}_2\; xs)\; x
\end{aligned}
$$

The master plan: BX + IP

Problem: Of the view-update laws

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

$$\texttt{get}\ (\texttt{put}\ xs\ ys) = ys$$

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

The master plan: BX + IP

Problem: Of the view-update laws

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

$$\texttt{get}\ (\texttt{put}\ xs\ ys) = ys$$

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

Like, for `get` $=$ `init`:

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

The master plan: BX + IP

Problem: Of the view-update laws

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

$$\texttt{get}\ (\texttt{put}\ xs\ ys) = ys$$

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

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

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

But then one would synthesize:

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

The master plan: BX + IP

Problem: Of the view-update laws

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

$$\texttt{get}\ (\texttt{put}\ xs\ ys) = ys$$

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

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

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

But then one would synthesize:

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

<u>1. possible solution:</u> Enforce use of both arguments?

The master plan: BX + IP

First, on a simpler example, $\texttt{get} = \texttt{head}$:

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

The master plan: BX + IP

First, on a simpler example, $\texttt{get} = \texttt{head}$:

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

To avoid $\texttt{put}\ xs\ y = xs$, insist on use of y, i.e., something like:

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

The master plan: BX + IP

First, on a simpler example, $\texttt{get} = \texttt{head}$:

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

To avoid $\texttt{put}\ xs\ y = xs$, insist on use of y, i.e., something like:

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

Starting from this hypothesis, practically only one reasonable path of synthesis, with result something like:

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

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

different "degrees" of use of *ys* in $\texttt{put}\; xs\; ys$ are possible.

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

different "degrees" of use of *ys* in $\texttt{put}\ xs\ ys$ are possible.

For example:

$$\texttt{put}\ xs\ ys = (\texttt{take}\ (\texttt{length}\ ys)\ xs) +\!\!+ [\texttt{last}\ xs]$$

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

different "degrees" of use of ys in $\texttt{put}\;xs\;ys$ are possible.

For example:

$$\texttt{put}\;xs\;ys = (\texttt{take}\;(\texttt{length}\;ys)\;xs) +\!\!+ [\texttt{last}\;xs]$$

Or:

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

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

different "degrees" of use of *ys* in $\texttt{put}\; xs\; ys$ are possible.

For example:

$$\texttt{put}\; xs\; ys = (\texttt{take}\; (\texttt{length}\; ys)\; xs) +\!\!+ [\texttt{last}\; xs]$$

Or:

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

Caution: Not every $\texttt{put}$ generated (like) above automatically satisfies $\texttt{get}\; (\texttt{put}\; xs\; ys) = ys$.

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

different "degrees" of use of *ys* in $\texttt{put}\; xs\; ys$ are possible.

For example:

$$\texttt{put}\; xs\; ys = (\texttt{take}\; (\texttt{length}\; ys)\; xs) \mathbin{+\!\!+} [\texttt{last}\; xs]$$

Or:

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

Caution: Not every $\texttt{put}$ generated (like) above automatically satisfies $\texttt{get}\; (\texttt{put}\; xs\; ys) = ys$. (But it's okay, trust IP.)

The master plan: BX + IP

2. possible solution: To after all generate I/O pairs for `put` from

$$\texttt{get}\ (\texttt{put}\ xs\ y) = y$$

as well, "inversion" of `get`.

The master plan: BX + IP

2. possible solution: To after all generate I/O pairs for `put` from

$$\texttt{get}\ (\texttt{put}\ xs\ y) = y$$

as well, "inversion" of `get`.

Then:

$$\texttt{put}\ xs\ y = \texttt{get}^{-1}\ y$$

as provider of further I/O pairs beside $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs$.

The master plan: BX + IP

2. possible solution: To after all generate I/O pairs for `put` from

$$\texttt{get}\ (\texttt{put}\ xs\ y) = y$$

as well, "inversion" of `get`.

Then:

$$\texttt{put}\ xs\ y = \texttt{get}^{-1}\ y$$

as provider of further I/O pairs beside $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs$.

Like, for $\texttt{get} = \texttt{head}$,

$$\texttt{head}^{-1}\ y = [y]$$

The master plan: BX + IP

2. possible solution: To after all generate I/O pairs for put from

$$\texttt{get}\ (\texttt{put}\ xs\ y) = y$$

as well, "inversion" of get.

Then:

$$\texttt{put}\ xs\ y = \texttt{get}^{-1}\ y$$

as provider of further I/O pairs beside $\texttt{put}\ xs\ (\texttt{get}\ xs) = xs$.

Like, for $\texttt{get} = \texttt{head}$,

$$\texttt{head}^{-1}\ y = [y]$$

or, better,

$$\texttt{head}^{-1}\ y = (y : \_)$$

The master plan: BX + IP

2. possible solution: To after all generate I/O pairs for put from

$$\texttt{get}\ (\texttt{put}\ \mathit{xs}\ y) = y$$

as well, "inversion" of get.

Then:

$$\texttt{put}\ \mathit{xs}\ y = \texttt{get}^{-1}\ y$$

as provider of further I/O pairs beside $\texttt{put}\ \mathit{xs}\ (\texttt{get}\ \mathit{xs}) = \mathit{xs}$.

Like, for $\texttt{get} = \texttt{head}$,

$$\texttt{head}^{-1}\ y = [y]$$

or, better,

$$\texttt{head}^{-1}\ y = (y : \_)$$

In this case, agrees with the other suggestion ...

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

The master plan: BX + IP

On the more complex example, $\texttt{get} = \texttt{init}$:

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

Use of

$$\texttt{init}^{-1}\ ys = \texttt{snoc}\ ys\ \_$$

The master plan: BX + IP

On the more complex example, $\mathtt{get} = \mathtt{init}$:

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

Use of

$$\mathtt{init}^{-1}\ ys = \mathtt{snoc}\ ys\ \_$$

to provide, beside:

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

also:

$$
\begin{aligned}
&\mathtt{put}\ [a]\quad\ [b] = [b, \_] \\
&\mathtt{put}\ [a, b]\ [\,]\ \ = [\_] \\
&\mathtt{put}\ [a, b]\ [c] = [c, \_] \\
&\ldots
\end{aligned}
$$

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
- Inductive Program Synthesis:
  - application of machine learning
  - detects/exploits regularities
  - hypothesis: captures 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
- Inductive Program Synthesis:
  - application of machine learning
  - detects/exploits regularities
  - hypothesis: captures programmer intentions
- Connection:
  - inductive program synthesis as a "helper"
  - 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;
    ...

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.