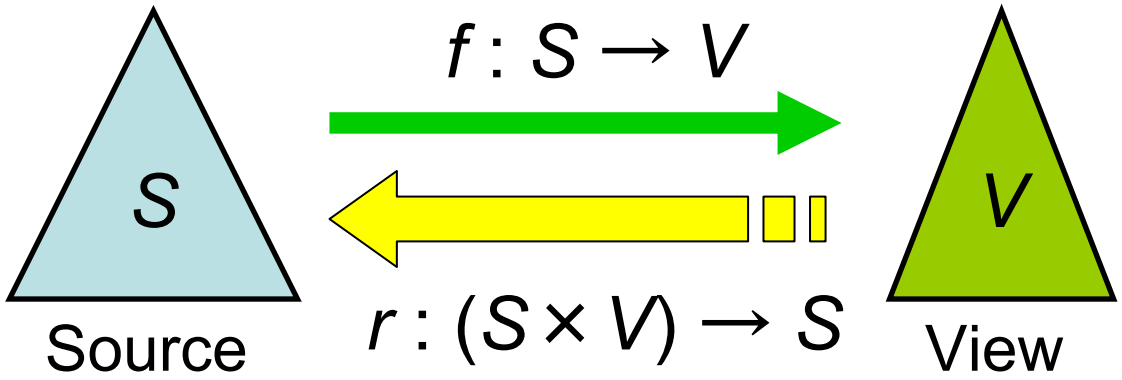
Bidirectionalization Transformation Based on Automatic Derivation of View Complement Functions

Kazutaka Matsuda*
Zhenjiang Hu*, Keisuke Nakano*
Makoto Hamana**, Masato Takeichi*

*The University of Tokyo **Gunma University

Bidirectional Transformation

- “Bidirectional Transformation”
  =“View Function” + “Backward Transformation”

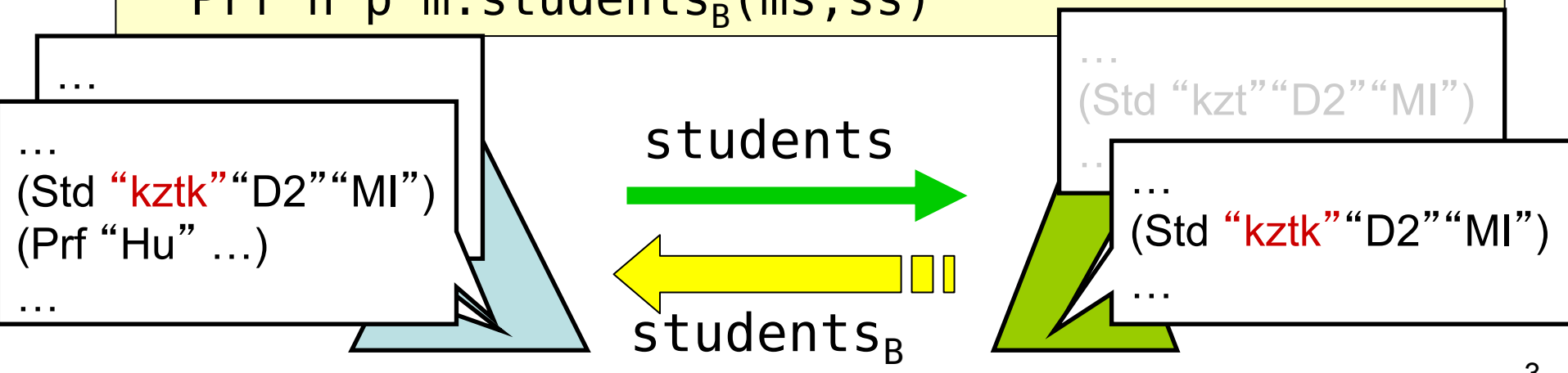


[Hegner 90] and so on

Example: students

```
students [] = []
students (Std n g m:ms) = Std n g m:students ms
students (Prf n p m:ms) = students ms

students_B([],[]) = []
students_B(Std n g m:ms, Std n' g' m':ss) =
  Std n' g' m':students_B(ms,ss)
students_B(Prf n p m:ms, ss) =
  Prf n p m:students_B(ms,ss)
```

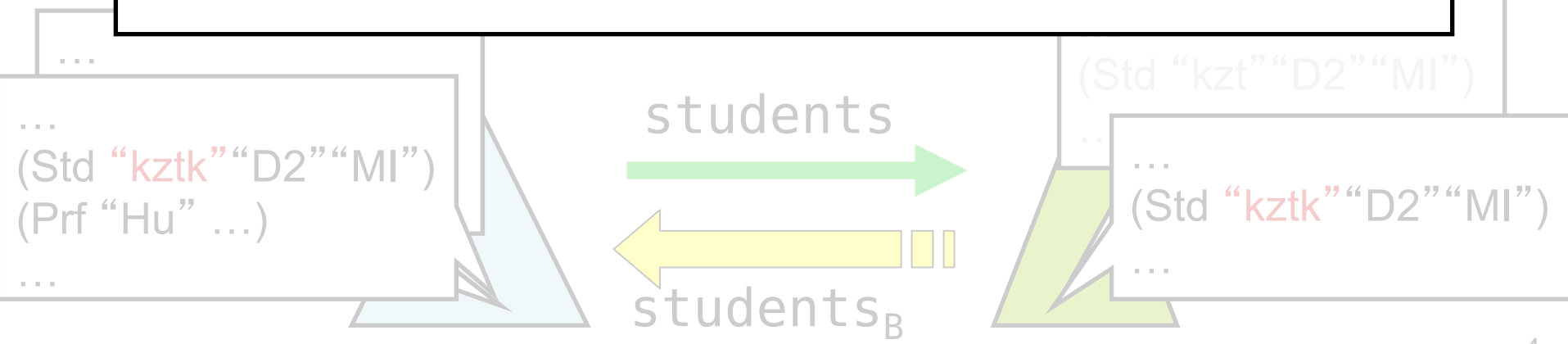


Example: students

```
students [] = []
students (Std n g m:ms) = Std n g m:students ms
students (Prf n p m:ms) = students ms
```

Problem

- Consistency of Both Programs
  - Backward program for changed forward program?



Existing Work

- Bidirectional Combinators [Foster et al. 05]
  - ☺ High expressive power thanks to combinators
  - ☹ Manually-defined basic transformations

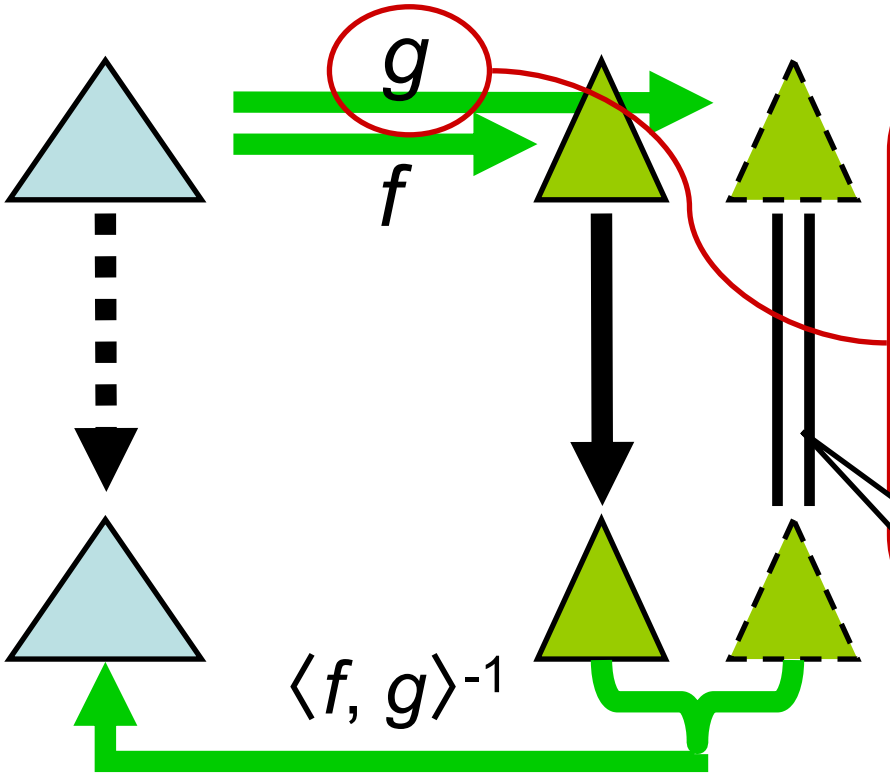
- Constant Complement Bidirectionalization [Bancilhon & Spyratos 81]
  - ☺ Automatic generation of backward transformations
  - ☹ No discussion of algorithms to derive complements
    - for RDB [Laurent et al. 01, Lechtenböger & Vossen 03]

Our Contribution

-- Combining Both Advantages --

- Bidirectionalization Transformation
  - for basic view functions
    - Affine and Treeless language
  - based on constant complement bidirectionalization
  - by 2 analyses and 3 program transformations
    - Range Analysis and Injectivity Analysis
    - Complement Derivation, Tupling and Inversion

Bidirectionalization Based on Constant-Complement



Complement Function

g is a complement function of *f* if ⟨*f*, *g*⟩ is injective.

※⟨*f*, *g*⟩(*x*) = (*f*(*x*),*g*(*x*))

g preserves lost information through transformation by *f*

Constant

Backward Transformation

$$r(s,v) = \langle f, g\rangle^{-1}(v, g(s))$$

[Bancilhon & Spyratos 81]

Backward Transformations using Complements

Complements of `fst(Pair(x,y)) = x`

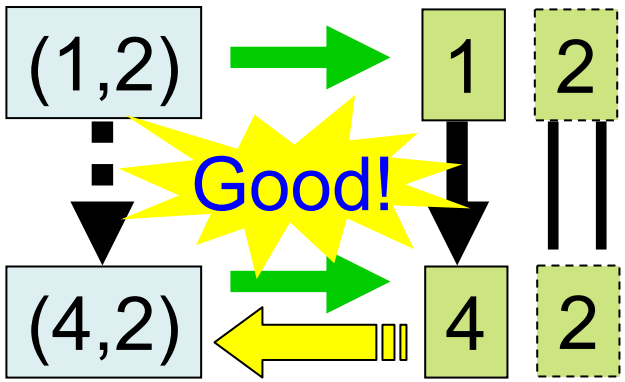
`snd(Pair(x,y)) = y`

`id(Pair(x,y)) = Pair(x,y)`

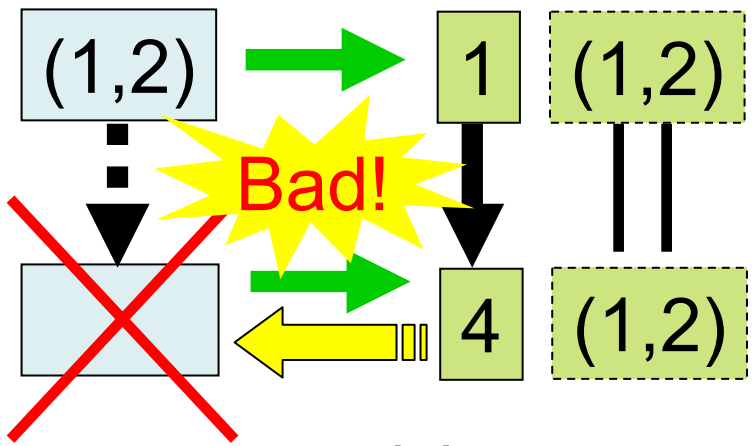
`...`

more than one

- A complement yields a backward transformation.

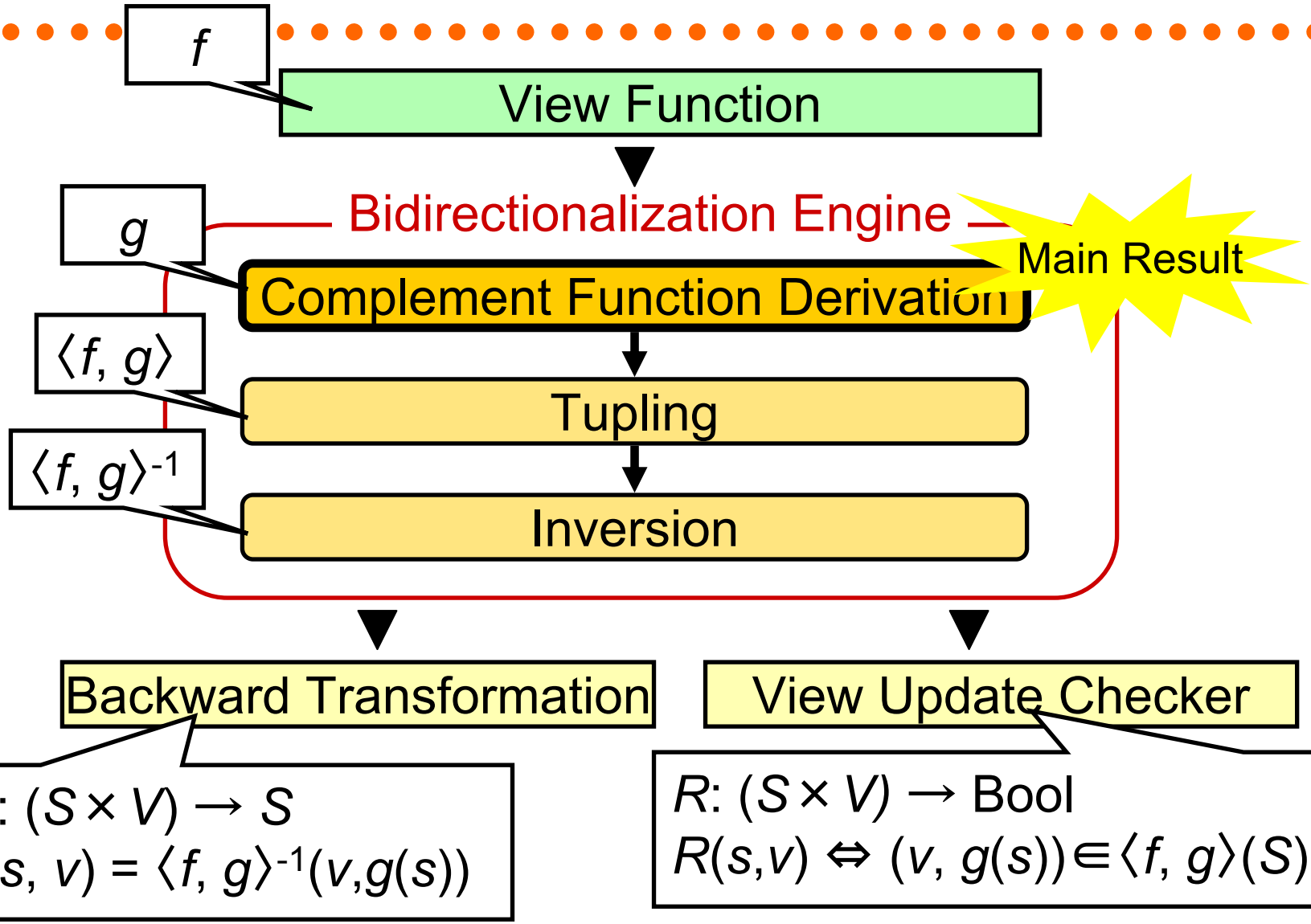


with `snd`

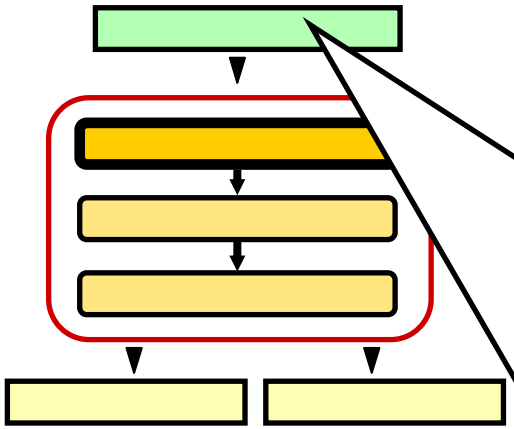


with `id`

Our System Overview



View Function

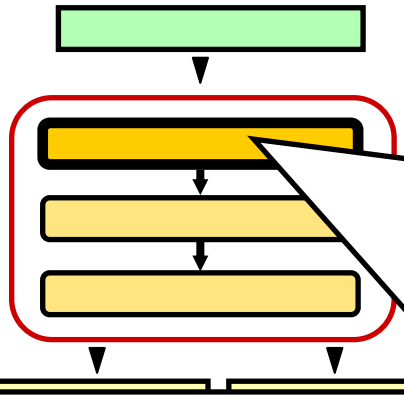


Affine and Treeless View Definition

```
students([]) = []
students(Std(n,g,m):ms) =
     Std(n,g,m):students(ms)
students(Prf(n,p,m):ms) =
     students(ms)
```

- Affine
  - Each variables is used at most once.
- Treeless [Wadler 90]
  - Arguments of function calls are only variables.

Complement Derivation



Keep Discarded Variables
Record Computation Paths

Complement Derivation

```
students([]) = []
students(Std(n,g,m):ms) =
     Std(n,g,m):students(ms)
students(Prf(n,p,m):ms) =
     students(ms)
```

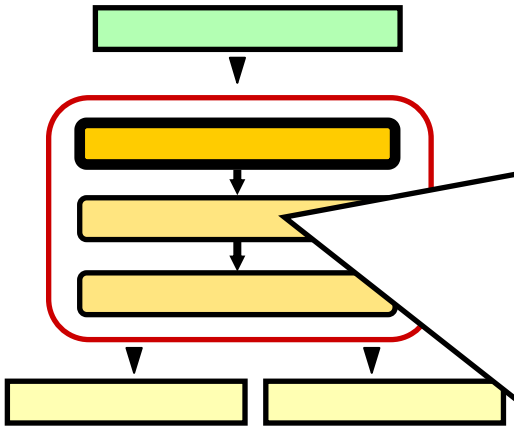
$\underline{\text{students}^c}([]) = B_1$

$\underline{\text{students}^c}(\text{Std}(n,g,m):ms) = B_2(\underline{\text{students}^c}(ms))$

$\underline{\text{students}^c}(\text{Prf}(n,p,m):ms) = B_3(n,p,m,\underline{\text{students}^c}(ms))$

Two Analyses:
- Range Analysis
- Injectivity Analysis

Tupling



Tupling

$f(p)=\ldots$ $f^c(p)=\ldots$

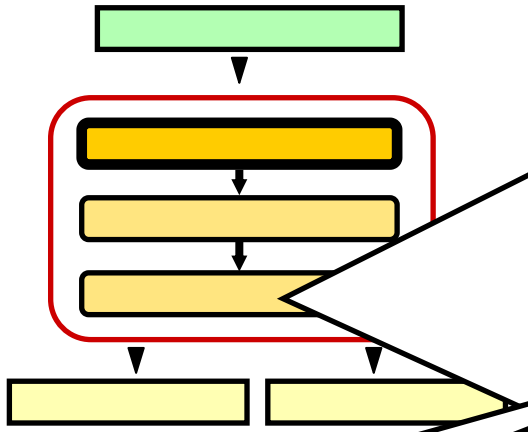
$f_\triangle(x) \equiv (f(x), f^c(x))$

```
students△([]) = ([],B1)
students△(Std(n,g,m):ms) =
   (Std(n,g,m):s, B2(t))
     where (s,t) = students△(ms)
students△(Prf(n,p,m):ms) =
   (s,B3(n,p,m,t))
     where (s,t) = students△(ms)
```

Form of function

```
f△(p)=q
  where x=g△(y)
```

Inversion



Non-deterministic Program

Inversion

$f_{\triangle}(p)=q$ **where** $x=g_{\triangle}(y)$

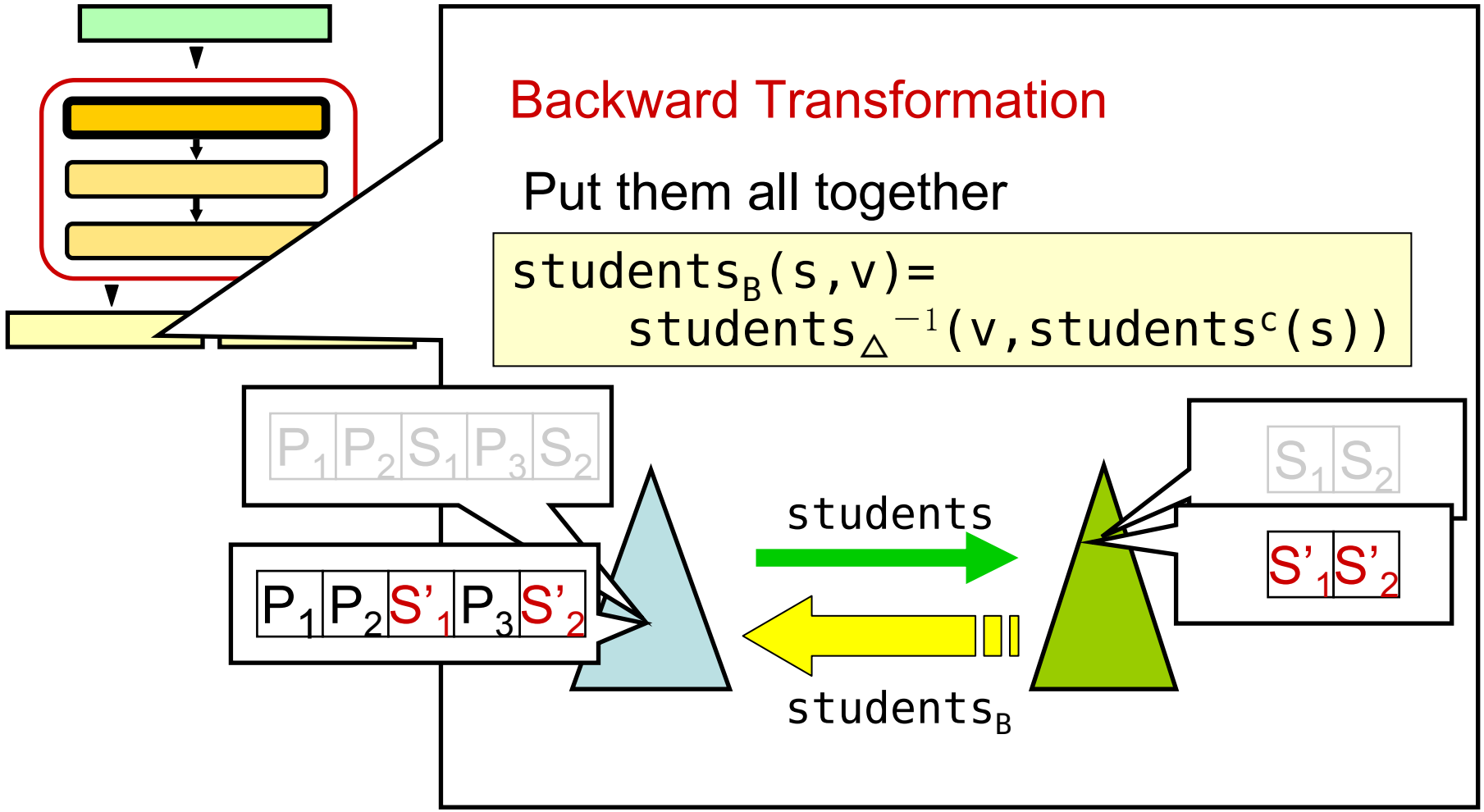
swap

$f_{\triangle}^{-1}(q)=p$ **where** $y=g_{\triangle}^{-1}(x)$

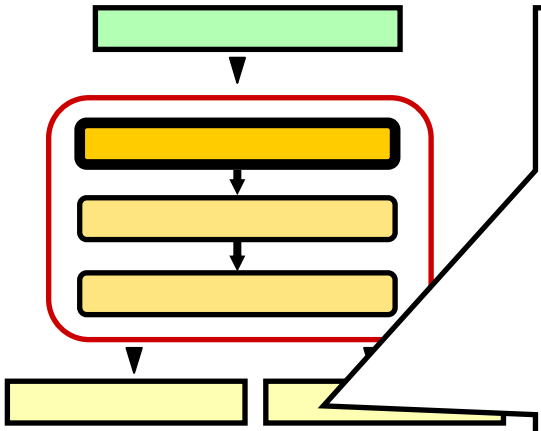
```
students△⁻¹([],B1) = []
students△⁻¹(Std(n,g,m):s,B2(t)) =
   Std(n,g,m):ms
     where ms = students△⁻¹(s,t)
students△⁻¹(s,B3(n,p,m,t)) =
   Prf(n,p,m):ms
     where ms = students△⁻¹(s,t)
```

Deterministic inverse for all the tested functions

Backward Transformation



View Update Checker



View Update Checker

Checking if an updated view can be reflected

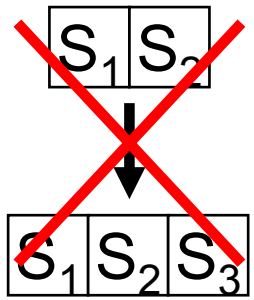
Suppose an initial source is

P_1	P_2	S_1	P_3	S_2

Bottom-up tree automaton

$\mathtt{Std}(q_{\text{any}}, q_{\text{any}}, q_{\text{any}}) : q_1 \rightarrow q_{\text{accept}}$
$\mathtt{Std}(q_{\text{any}}, q_{\text{any}}, q_{\text{any}}) : q_2 \rightarrow q_1$
$[\,] \rightarrow q_2$

It prohibits invalid view updates.



Conclusion

- Bidirectionalization transformation
  - based on derivation of complement functions
  - for affine and treeless programs
- View update checker
- System implementation
  - http://www.ipl.t.u-tokyo.ac.jp/~kztk/bidirectionalization/
  - Functions that we have tested in the system
    - fst, snd, half, add, max, min, zip, append, map, filter

Future Work

- Widening class of language
  - Duplications and function compositions
- Applying derivation principles to practical examples
  - XML transformations