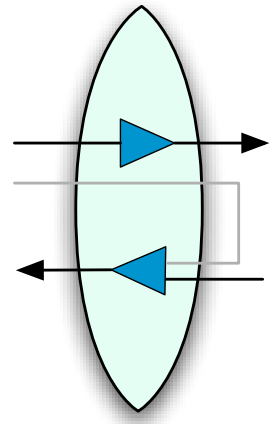
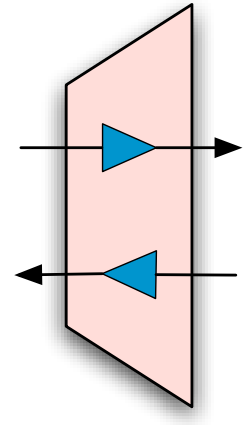
Quotient Lenses

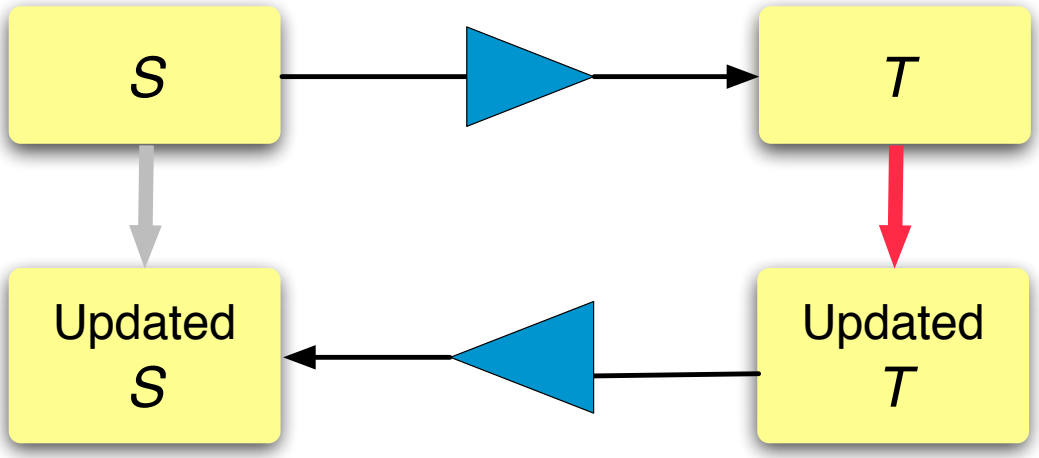


Nate Foster (Penn)
Benjamin C. Pierce (Penn)
Alexandre Pilkiewicz (Polytechnique/INRIA)

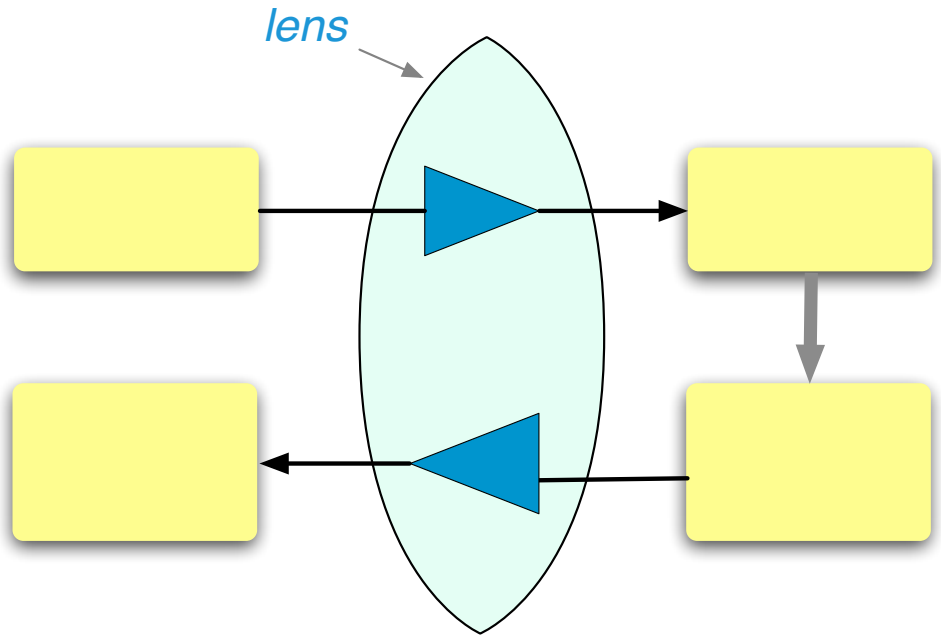


IBM PLDay '08

Bidirectional Transformations



Bidirectional Programming Language



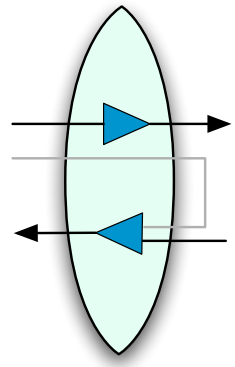
Eliminates Redundancy: programs describes two functions

Ensures Correctness: type system guarantees well-behavedness

Semantics

A lens l from S to T is a triple of functions

$$\begin{aligned} l.get &\in S \to T \\ l.put &\in T \to S \to S \\ l.create &\in T \to S \end{aligned}$$

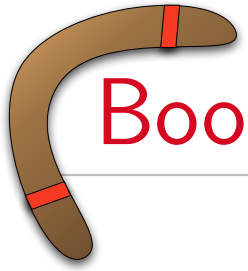


obeying three "round-tripping" laws:

$$l.put\ (l.get\ s)\ s = s \qquad (\textsc{GetPut})$$

$$l.get\ (l.put\ t\ s) = t \qquad (\textsc{PutGet})$$

$$l.get\ (l.create\ t) = t \qquad (\textsc{CreateGet})$$



Boomerang [POPL '08]

Data model: strings

Core combinators: finite-state transducers

Host language: λ-calculus, regular types, dependent types, user-defined data types, polymorphism

Lenses: addresses books, bibliographies, CSV, documents, scientific data, XML

Applications: converters, synchronizers, structure editors

Example: MediaWiki (Get)

```
==Chefs==
* Julia Child
* Jacques Pepin
==Justices==
* Warren Burger
* Arthur Goldberg
```

```
<html>
 <body>
  <h2>Chefs</h2>
  <ul>
   <li>Julia Child</li>
   <li>Jacques Pepin</li>
  </ul>
  <h2>Justices</h2>
  <ul>
   <li>Warren Burger</li>
   <li>Arthur Goldberg</li>
  </ul>
 </body>
</html>
```

Example: MediaWiki (Put)

```
==Chefs==
* Julia Child
==Justices==
* Arthur Goldberg
```

```
<html>
 <body>
  <h2>Chefs</h2>
  <ul>
   <li>Julia Child</li>
  </ul>
  <h2>Justices</h2>
  <ul>
   <li>Arthur Goldberg</li>
  </ul>
 </body>
</html>
```

Example: MediaWiki (Lens)

```
(* helpers *)
let mk_elt (ws:string) (tag:string) (body:lens) = ...
let mk_simple_elt (ws:string) (tag:string) (body:lens) =
  qins WS ws .
  ins ("<" . tag . ">") .
  body .
  ins ("</" . tag . ">")

(* main lenses *)
let p : lens =
  mk_simple_elt nl4 "p" ((text . nl)* . (text . del nl))
let li : lens =
  mk_simple_elt nl6 "li" (del "* " . text)
let ul : lens =
  mk_elt nl4 "ul" (li . del nl)+
let h2 : lens =
  mk_simple_elt nl4 "h2" (del "==" . text . del "==")
let s : lens =
  (del nl . (p | ul))*
let html : lens =
  mk_outer_elt nl0 "html" (mk_elt nl2 "body" s* )
```

This Talk: Lenses for... ?

This Talk: Lenses for Whitespace!

Many data formats contain inessential information:

```
<html>\n
__<body>\n
____<h2>Famous Chefs</h2>\n
____<ul>\n
______<li>Julia Child</li>\n
____</ul>\n
____<h2>Supreme Court Justices</h2>\n
____<ul>\n
______<li>Arthur Goldberg</li>\n
____</ul>\n
__</body>\n
</html>\n
```

This Talk: Lenses for Whitespace!

Many data formats contain inessential information:

```
<html>\n
<body>\n
<h2>Famous Chefs</h2>\n
<ul>\n
<li>Julia Child</li>\n
</ul>\n
<h2>Supreme Court Justices</h2>\n
<ul>\n
<li>Arthur Goldberg</li>\n
</ul>\n
</body>\n
</html>\n
```

This Talk: Lenses for Whitespace!

Many data formats contain inessential information:

```
<html><body>\n
__<h2>Famous Chefs</h2>\n
__<ul><li>Julia Child</li></ul>\n
__<h2>Supreme Court Justices</h2>\n
__<ul><li>Arthur Goldberg</li></ul>\n
</body></html>\n
```

Want the *put* function to treat these views equivalently but

$$l.get\ (l.put\ t\ s) = t \qquad \text{(PutGet)}$$

implies they must map to different sources!

Dealing With Ignorable Data

Approach #1: No laws.

Transformations not required to obey any formal properties.

But clearly intended to be "essentially" bidirectional.

Backed up by intuitive understanding of implementation.

Examples:

- biXid [Kawanaka and Hosoya '06]
- PADS [AT&T / Princeton]

Dealing With Ignorable Data

Approach #2: Weaker laws.

Replace round-trip laws with round-trip-and-a-half versions.

Allows transformations that normalize data in the view...

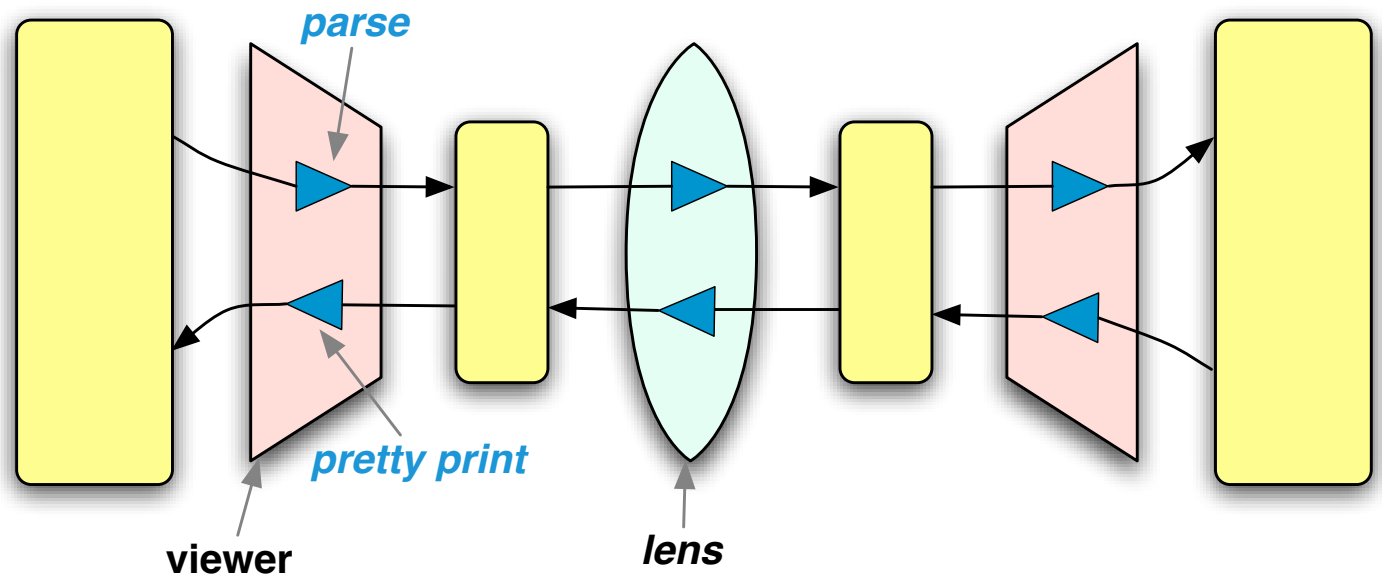
...along with a whole host of ill-behaved transformations.

Examples:

- Inv [Mu,Hu,Takeichi '04]
- X [Hu,Mu,Takeichi '04]
- Bi-XQuery [Liu, Hu, Takeichi '07]

Dealing With Ignorable Data

Approach #3: Viewers.



Examples:

- Focal [POPL '05]
- XSugar [Brabrand, Møller, Schwartzbach '05]

Dealing With Ignorable Data

Or... develop a theory of lenses that are well-behaved modulo equivalence relations on the source ($\sim_S$) and target ($\sim_T$).

Dealing With Ignorable Data

Or... develop a theory of lenses that are well-behaved modulo equivalence relations on the source ($\sim_S$) and target ($\sim_T$).

A quotient lens l satisfies the following laws

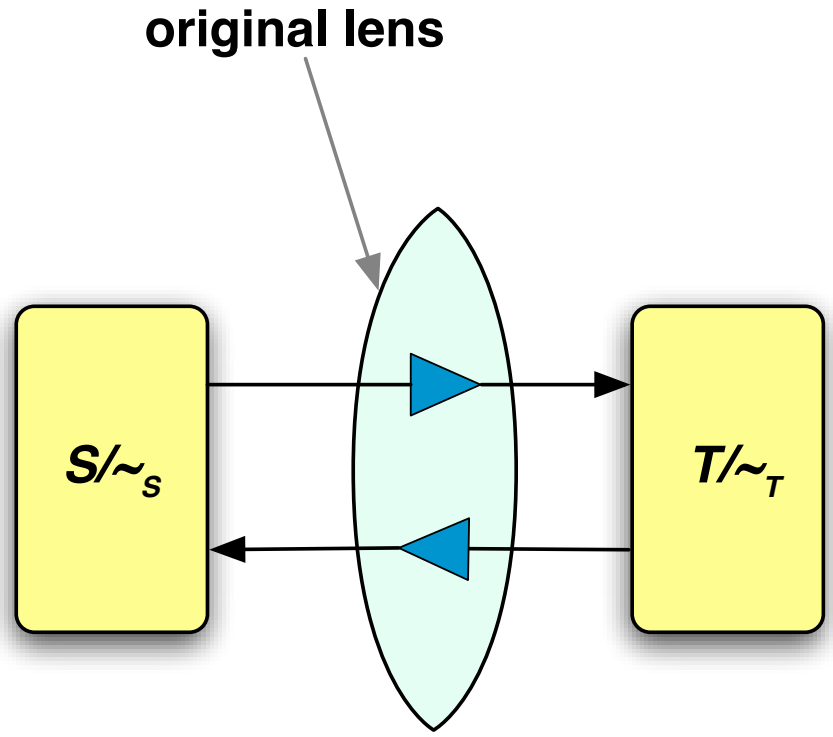
$$l.put\ (l.get\ s)\ s \sim_S s \qquad (\textsc{GetPut})$$

$$l.get\ (l.put\ t\ s) \sim_T t \qquad (\textsc{PutGet})$$

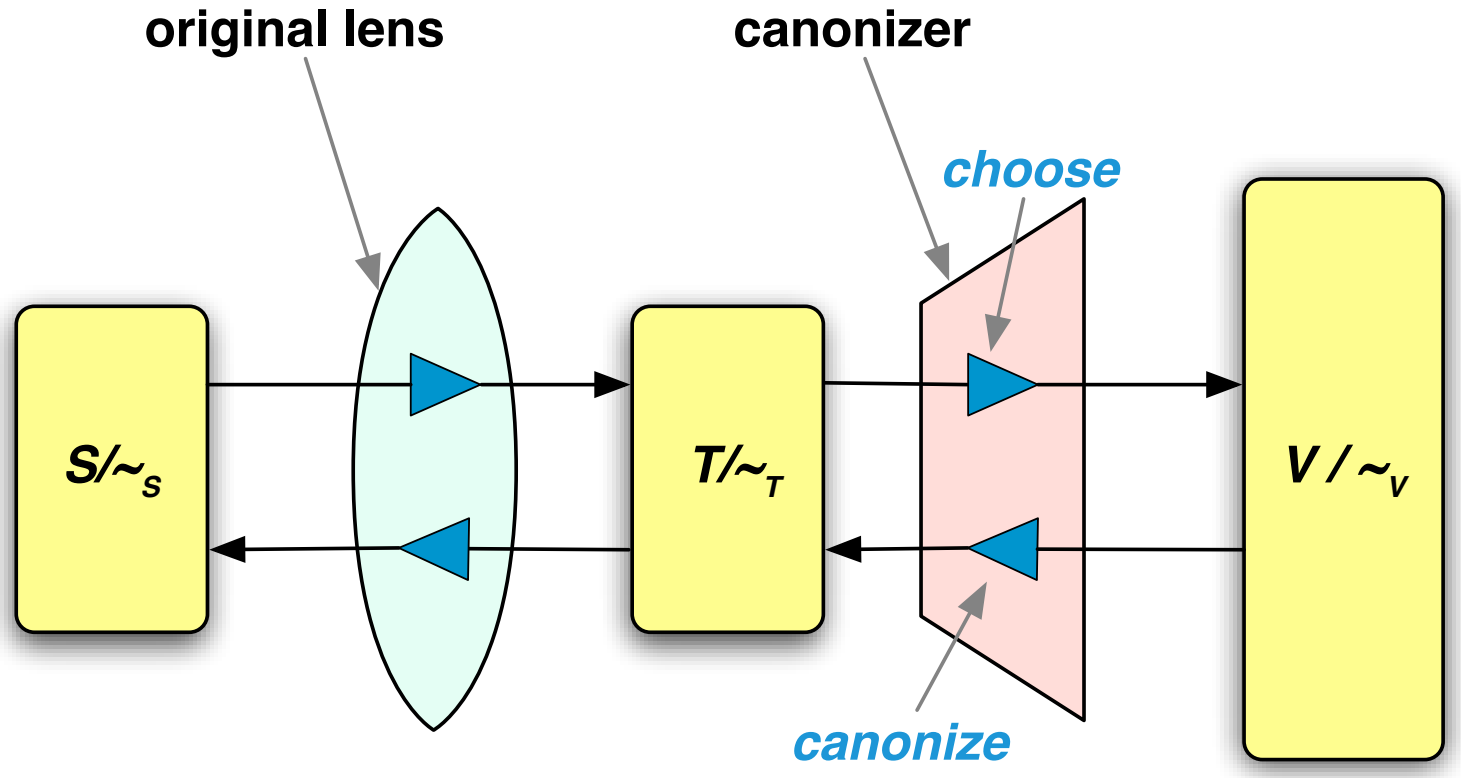
$$l.get\ (l.create\ t) \sim_T t \qquad (\textsc{CreateGet})$$

(Plus a few natural laws ensuring that the components of lenses respect $\sim_S$ and $\sim_T$.)

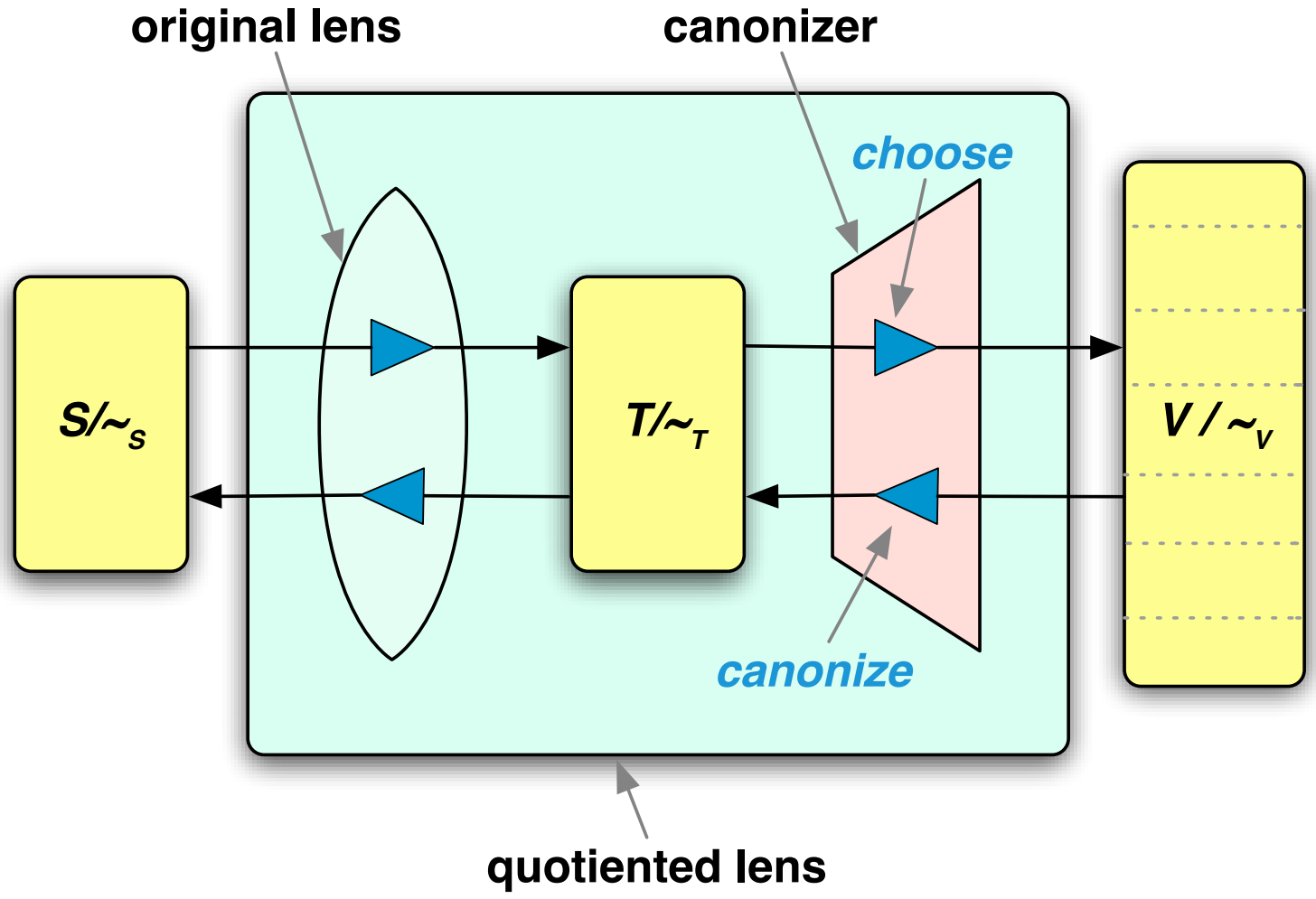
Quotient Lenses



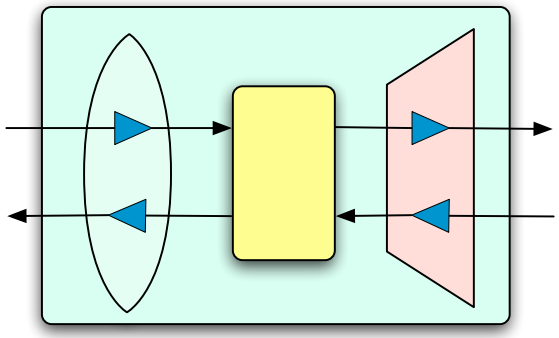
Quotient Lenses



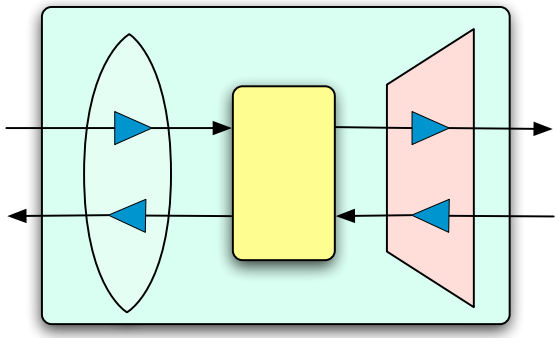
Quotient Lenses

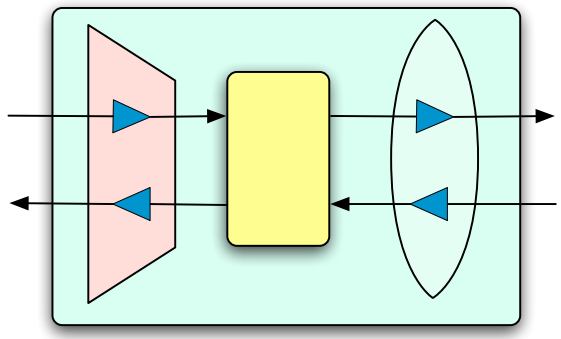


Quotient Lenses

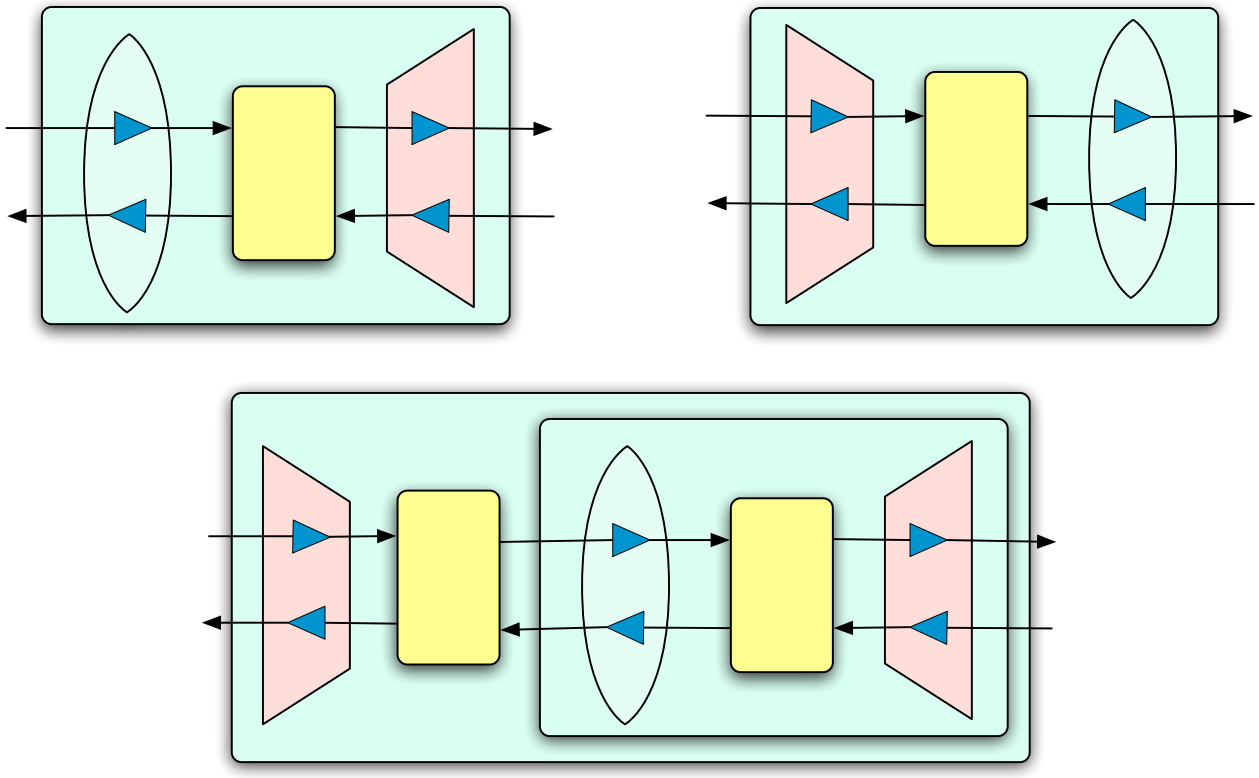


Quotient Lenses

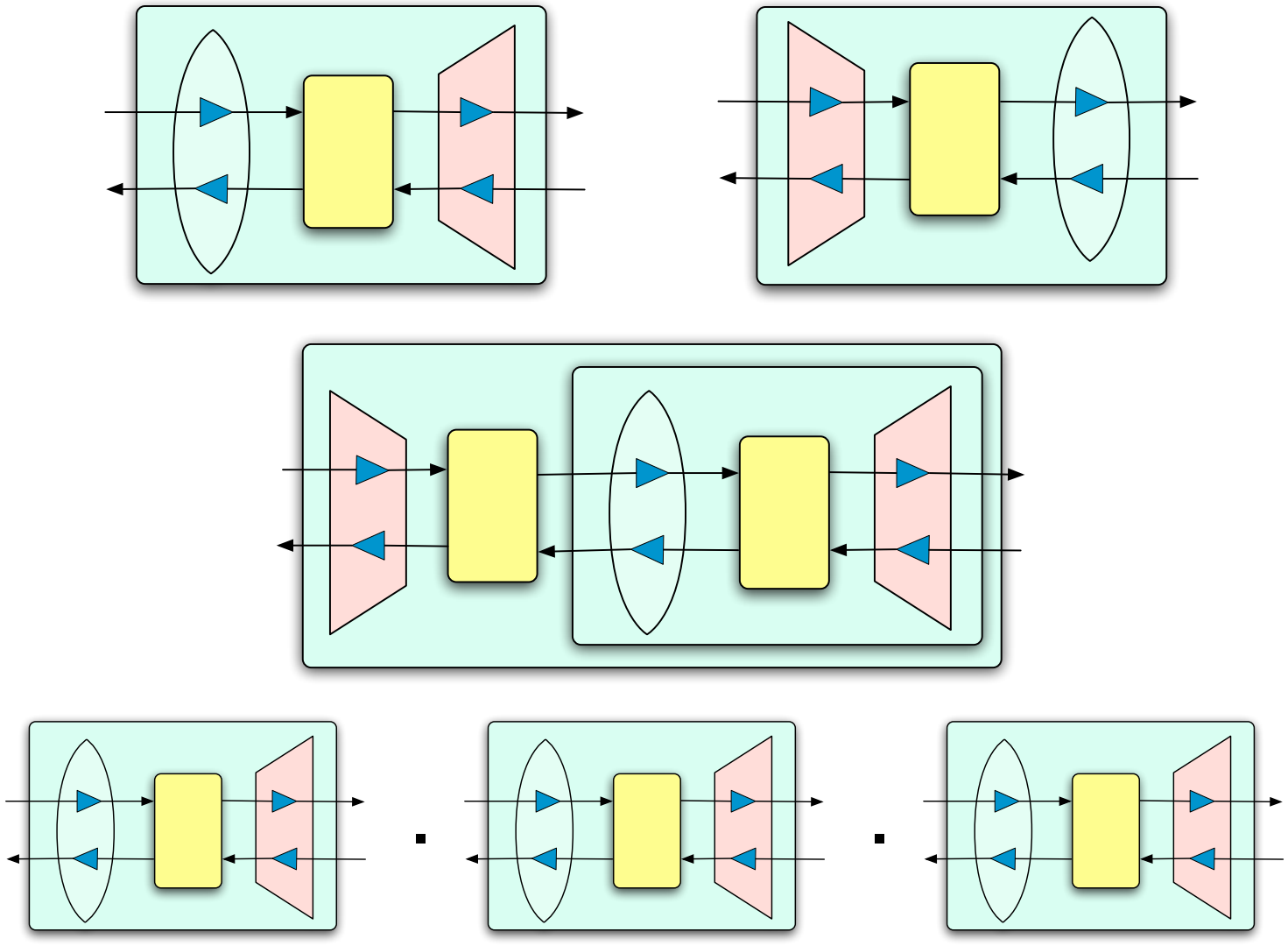




Quotient Lenses



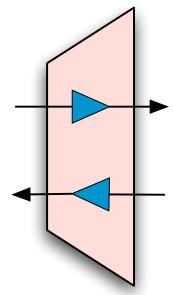
Quotient Lenses



Canonizers

A canonizer q from V to T is a pair of functions

$$\begin{aligned} q.canonize &\in V \to T \\ q.choose &\in T \to V \end{aligned}$$



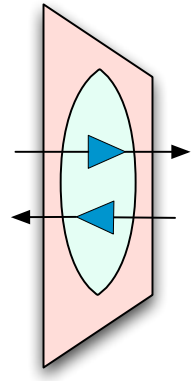
obeying just one law:

$$l.canonize\ (l.choose\ t)\ t = t \qquad \text{(ReCanonize)}$$

Syntax for Canonizers

Every lens l from V to T can be converted to a canonizer:

$$q.canonize \triangleq l.get$$
$$q.choose \triangleq l.create$$



The CreateGet law for l implies ReCanonize.

Additionally, the relaxed canonizer law enable primitives that are not valid as lenses:

- sorting
- duplicating
- wrapping long lines of text

Conclusion

The need to handle inessential data arises in many real-world applications built using lenses.

Much of this data is simple, but failing to deal with it renders many lenses essentially useless.

Quotient lenses are a critical piece of technology that helps bridge the gap between the theory and practice of bidirectional programming languages.

Canonizers lead to elegant syntax for quotient lenses.

Thank You!

Collaborators: Benjamin Pierce, Alexandre Pilkiewcz.

Other Boomerang contributors: Aaron Bohannon, Michael Greenberg, and Alan Schmitt.

Want to play? Boomerang is available for download:

- Source code (LGPL)
- Binaries for OS X, Linux
- Research papers
- Tutorial and growing collection of demos

`http://www.seas.upenn.edu/~harmony/`