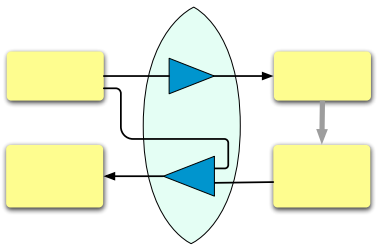
Boomerang: Resourceful Lenses for String Data

Aaron Bohannon (Penn)
J. Nathan Foster (Penn)
Benjamin C. Pierce (Penn)
Alexandre Pilkiewicz (École Polytechnique)
Alan Schmitt (INRIA)

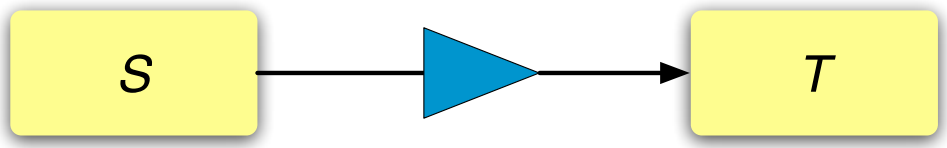
POPL '08



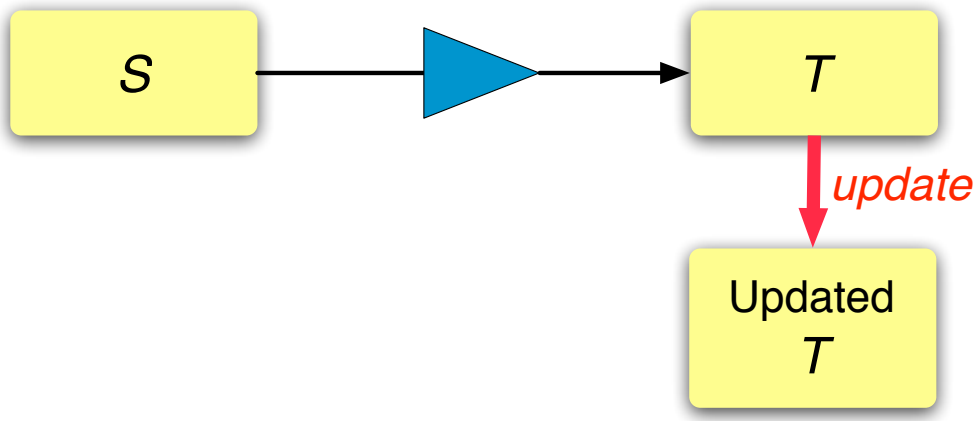
HARMONY

PENN PL

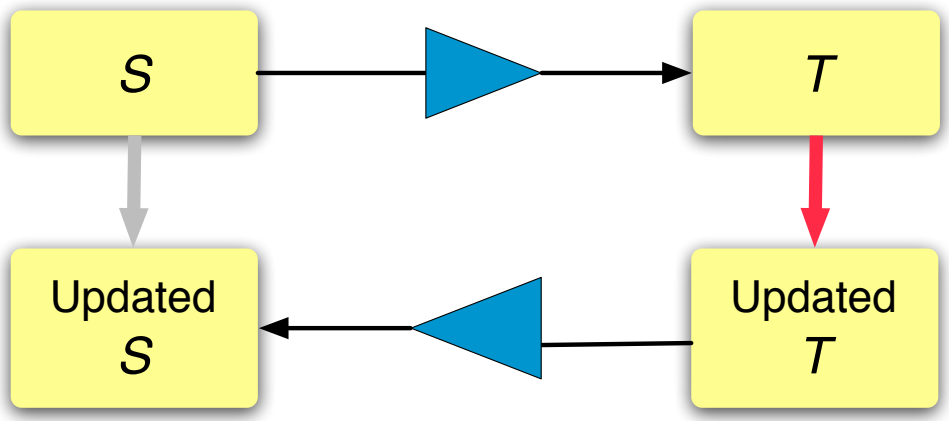
Bidirectional Mappings



Bidirectional Mappings

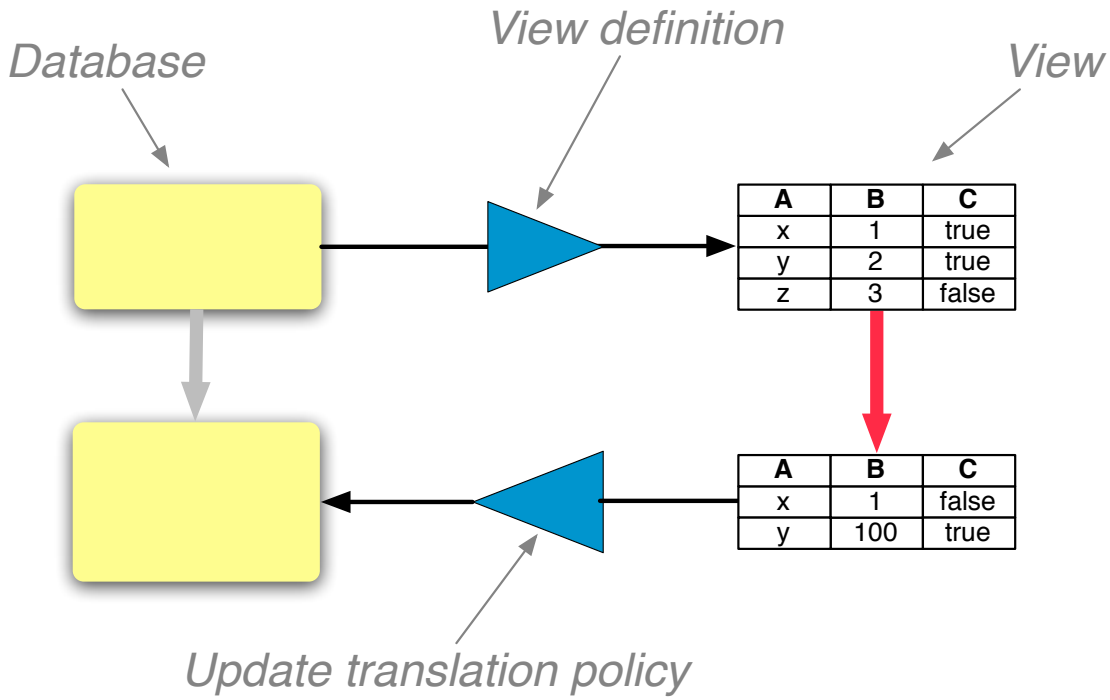


Bidirectional Mappings



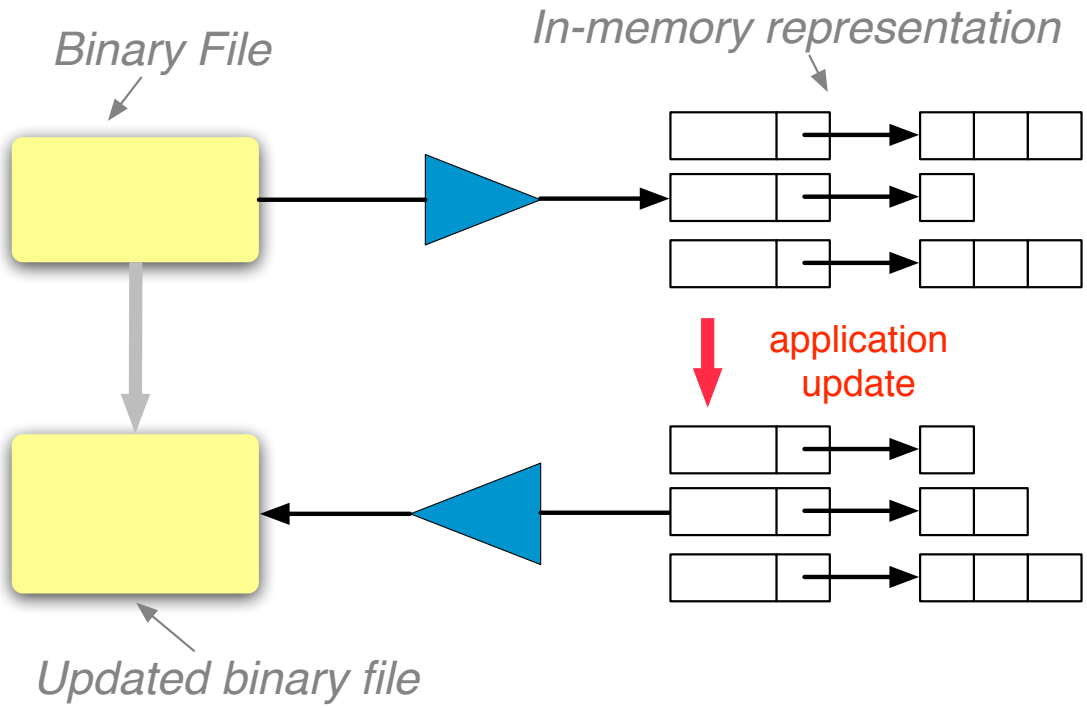
The View Update Problem

This is called the view update problem in the database literature.



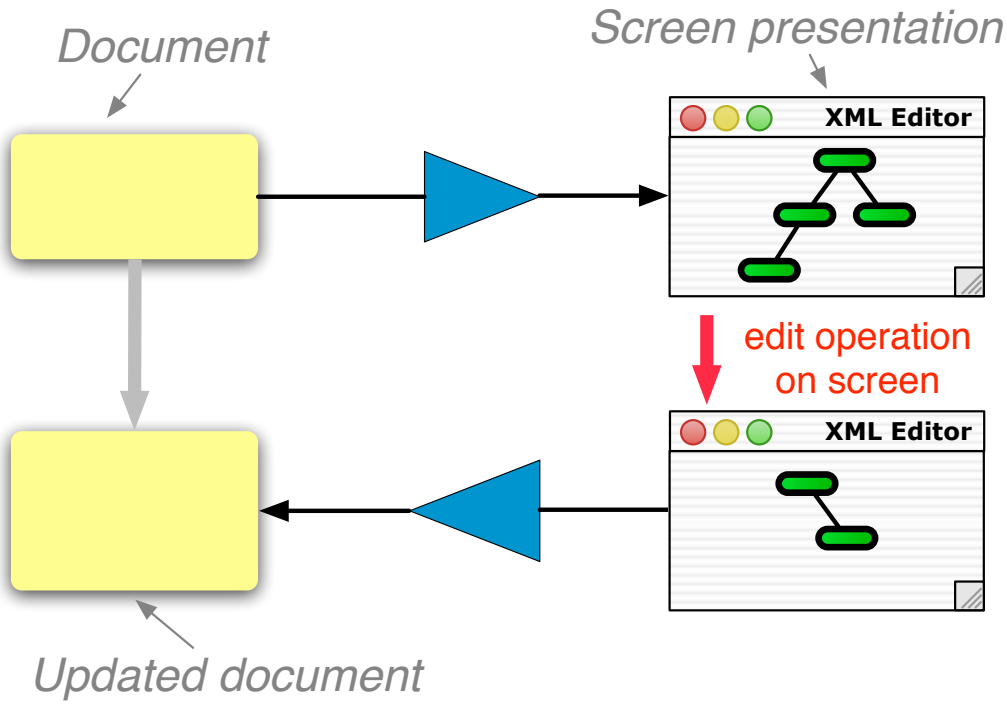
The View Update Problem In Practice

It also appears in picklers and unpicklers...



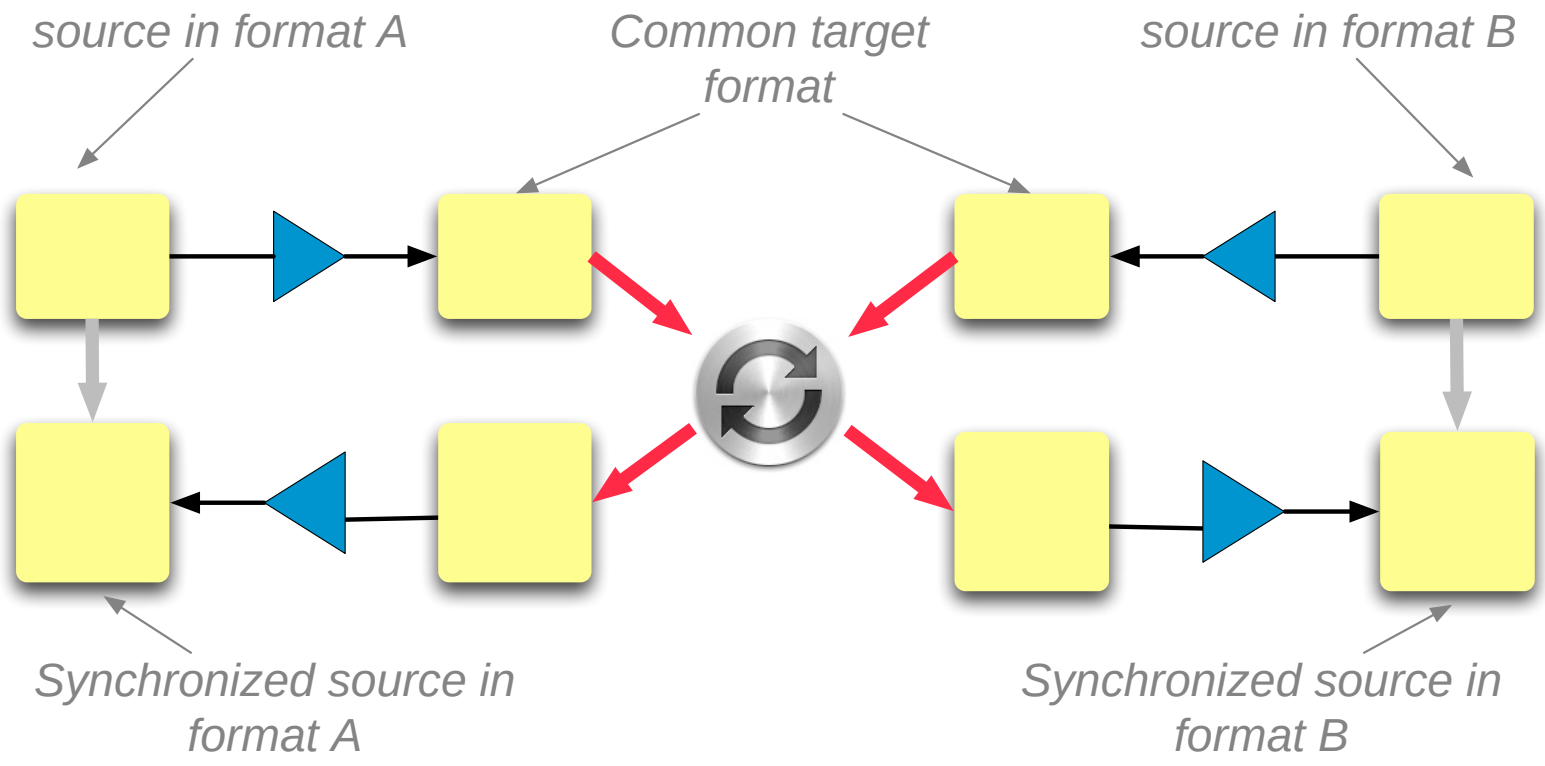
The View Update Problem In Practice

...in structure editors...

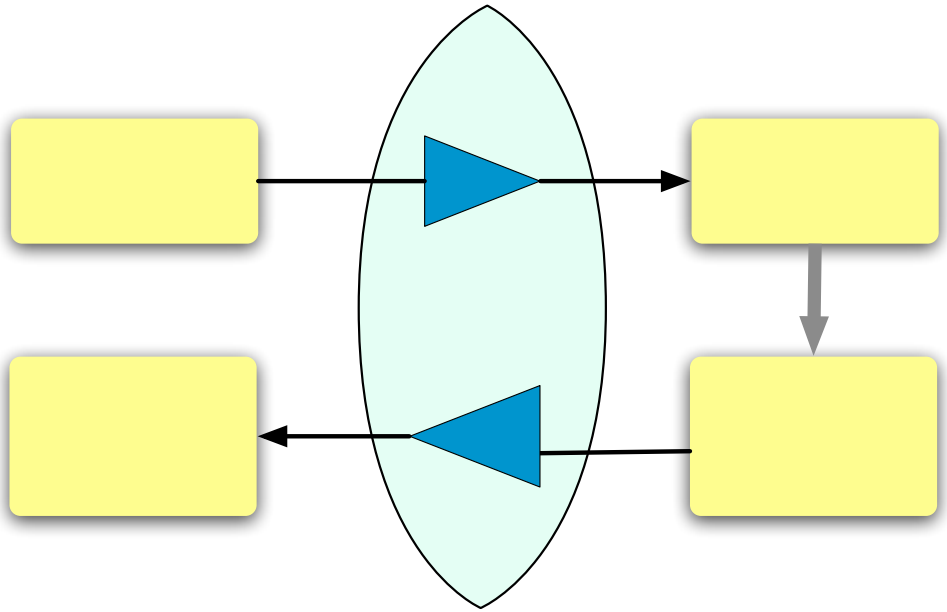


The View Update Problem In Practice

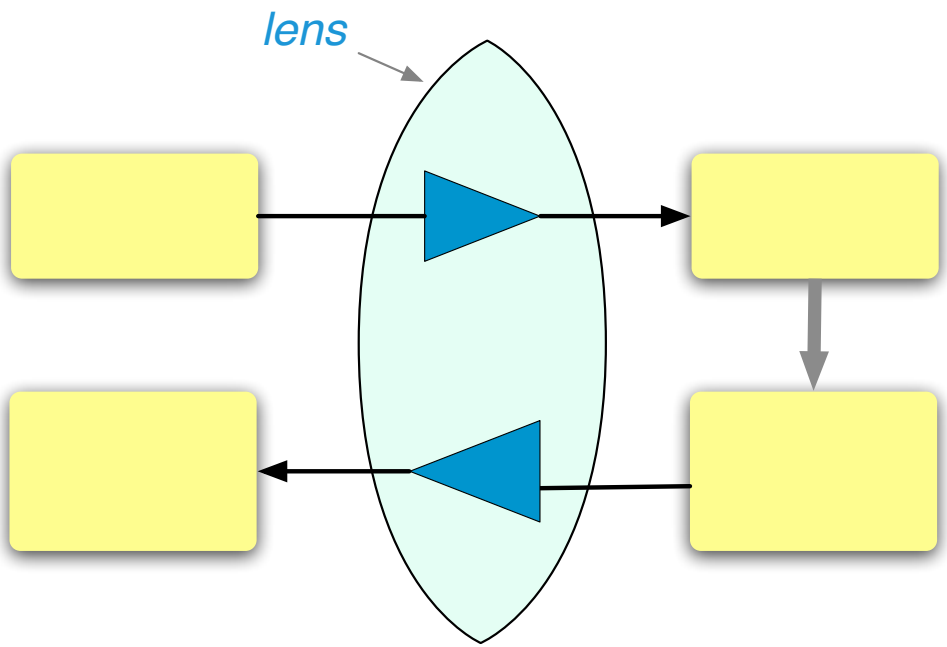
...and in data synchronizers like the Harmony system.



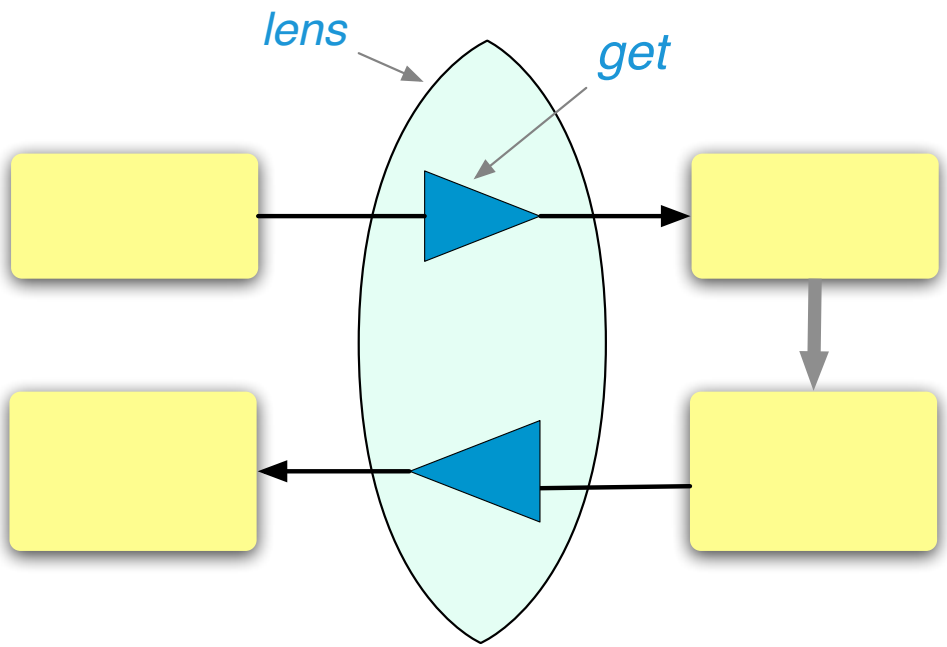
Linguistic Approach



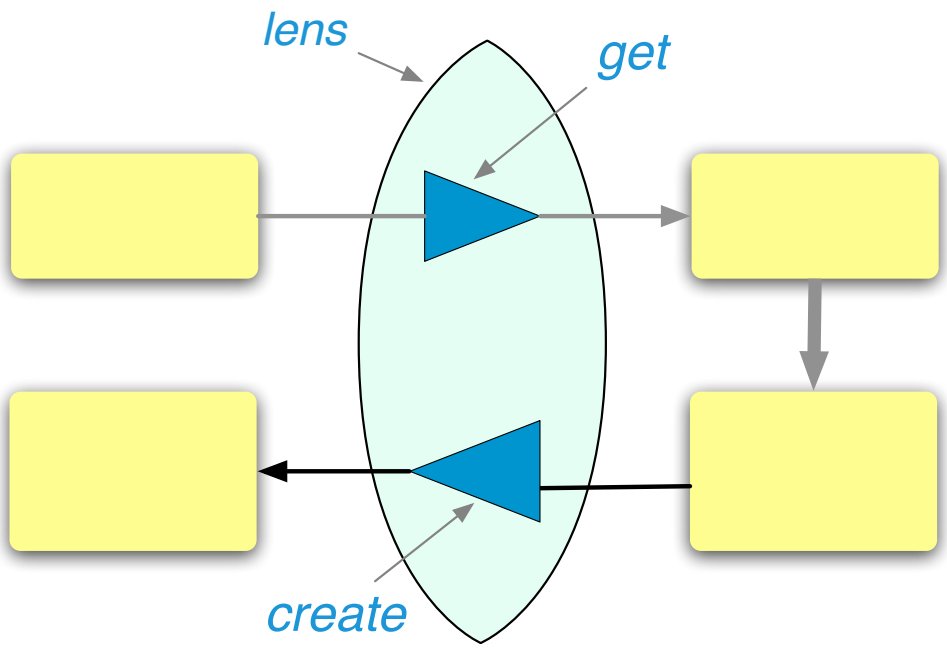
Terminology



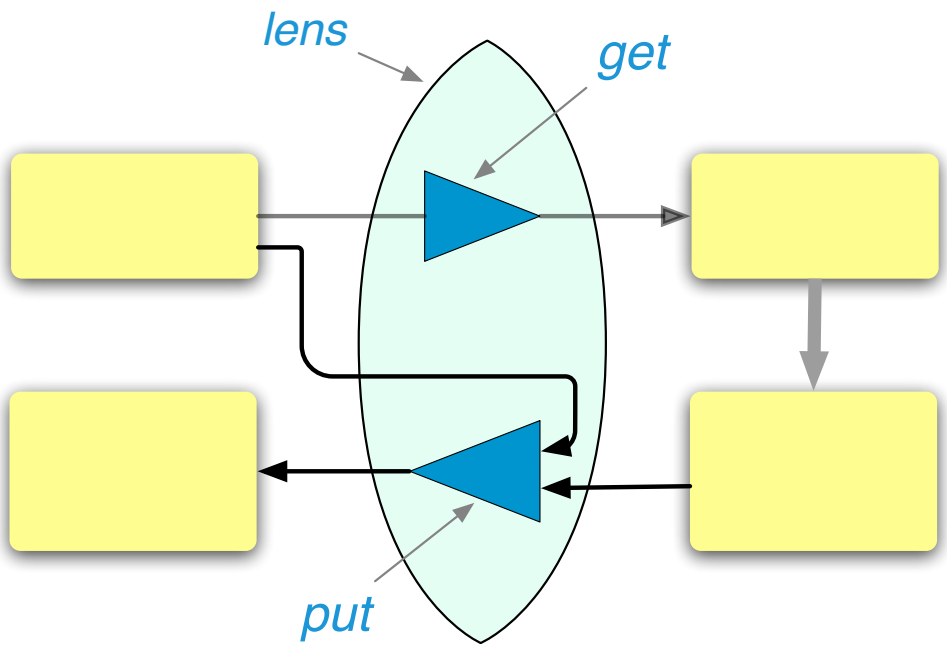
Terminology



Terminology



Terminology



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:

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

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

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

This Talk: Lenses for Ordered Data

Data model: Strings

Computation model: Finite-state transducers

Type system: Regular languages

Why strings?

- Simplest form of *ordered data*.
- There's a lot of string data in the world.

Contributions

String lenses: interpret finite-state transducers as lenses.

Dictionary lenses: refinement to handle problems with ordered data.

Boomerang: full-blown programming language built around core combinators.

Applications: lenses for real-world data formats.

Composer Lens (Get)

Source string:

```
"Benjamin Britten, 1913-1976, English"
```

Target string:

```
"Benjamin Britten, English"
```

Composer Lens (Get)

Source string:

```
"Benjamin Britten, 1913-1976, English"
```

Target string:

```
"Benjamin Britten, English"
```

Updated target string:

```
"Benjamin Britten, British"
```

Composer Lens (Put)

Putting new target

```
"Benjamin Britten, British"
```

into original source

```
"Benjamin Britten, 1913-1976, English"
```

yields new source:

```
"Benjamin Britten, 1913-1976, British"
```

Composer Lens (Definition)

```
let ALPHA : regexp = [A-Za-z ]+
let YEAR  : regexp = [0-9]{4}
let YEARS : regexp = YEAR . "-" . YEAR

let c : lens = cp ALPHA . cp ", "
             . del YEARS . del ", "
             . cp ALPHA
```

Benjamin Britten, 1913-1976, English

Benjamin Britten, English

Composers (Get)

Now let us extend the lens to handle ordered lists of composers — i.e., so that

```
"Aaron Copland, 1910-1990, American
 Benjamin Britten, 1913-1976, English"
```

maps to

```
"Aaron Copland, American
 Benjamin Britten, English"
```

Composers (Lens)

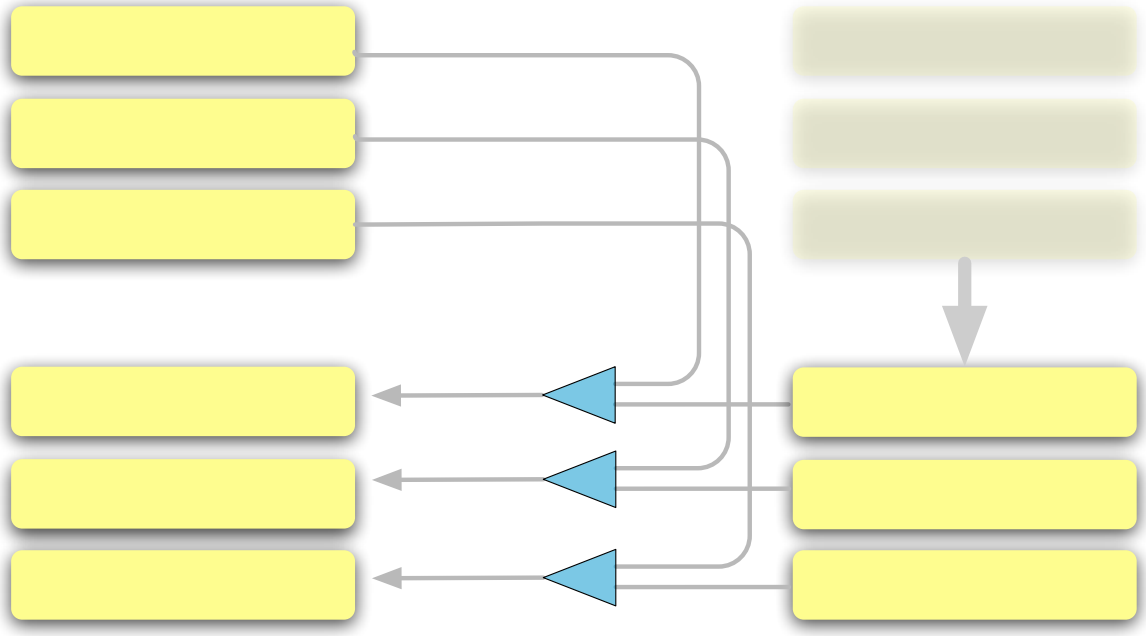
```
let ALPHA : regexp = [A-Za-z ]+
let YEAR  : regexp = [0-9]4
let YEARS : regexp = YEAR . "-" . YEAR

let c : lens = cp ALPHA . cp ", "
             . del YEARS . del ", "
             . cp ALPHA

let cs : lens = cp "" | c . (cp "\n" . c)*
```

Kleene-* and Alignment

Unfortunately, there is a serious problem lurking here.



A *put* function that works by position does not always give us what we want!

A Bad Put

Updating

```
"Aaron Copland, American
Benjamin Britten, English"
```

to

```
"Benjamin Britten, English
 Aaron Copland, American"
```

A Bad Put

... and then putting

```
"Benjamin Britten, English
 Aaron Copland, American"
```

into the same input as above...

```
"Aaron Copland, 1910-1990, American
 Benjamin Britten, 1913-1976, English"
```

...yields a mangled result:

```
"Benjamin Britten, 1910-1990, English
 Aaron Copland, 1913-1976, American"
```

This problem is *serious* and *pervasive*.

A Way Forward

In the composers lens, we want the *put* function to match up lines with identical name components. It should never pass

```
"Benjamin Britten, English"
```

and

```
"Aaron Copland, 1910-1990, American"
```

to the same *put*!

To achieve this, the lens needs to identify:

- where are the re-orderable chunks in source and target;
- how to compute a key for each chunk.

A Better Composers Lens

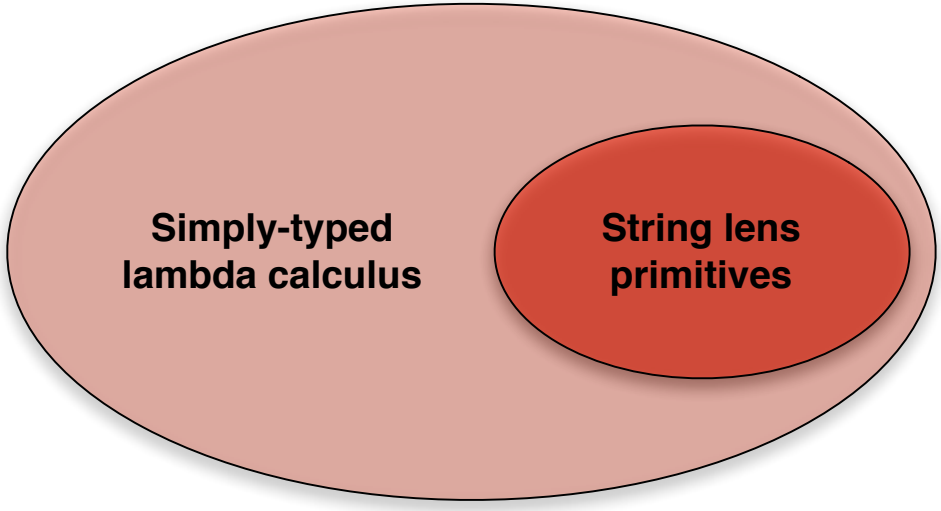
Similar to previous version but with a `key` annotation and a new combinator (`<c>`) that identifies the pieces of source and target that may be reordered.

```
let c = key ALPHA . cp ", "
      . del YEARS . del ", "
      . cp ALPHA
let cs = cp "" | <c> . (cp "\n" . <c>)*
```

The *put* function operates on a dictionary structure where source chunks are accessed by key.

Boomerang

Boomerang is a simply typed functional language over the base types `string`, `regexp`, `lens`, ...



Hybrid type checker [Flanagan, Freund et. al].

Demo

Bibliographic Data (BibTeX Source)

```
@inproceedings{utts07,
   author = {J. Nathan Foster
             and Benjamin C. Pierce
             and Alan Schmitt},
   title = {A {L}ogic {Y}our {T}ypechecker {C}an {C}ount {O}n:
            {U}nordered {T}ree {T}ypes in {P}ractice},
   booktitle = {PLAN-X},
   year = 2007,
   month = jan,
   pages = {80--90},
   jnf = "yes",
   plclub = "yes",
}
```

Bibliographic Data (RIS Target)

```
TY - CONF
ID - utts07
AU - Foster, J. Nathan
AU - Pierce, Benjamin C.
AU - Schmitt, Alan
T1 - A Logic Your Typechecker Can Count On:
     Unordered Tree Types in Practice
T2 - PLAN-X
PY - 2007/01//
SP - 80
EP - 90
M1 - jnf: yes
M1 - plclub: yes
ER -
```

Genomic Data (SwissProt Source)

```
CC -!- INTERACTION: Self;
   NbExp=1; IntAct=EBI-1043398, EBI-1043398;
   Q8NBH6:-;
   NbExp=1;
   IntAct=EBI-1043398, EBI-1050185;
   P21266:GSTM3;
   NbExp=1;
   IntAct=EBI-1043398, EBI-350350;
```

Genomic Data (UniProtKB Target)

```
<comment type="interaction">
   <interactant intactId="EBI-1043398"/>
   <interactant intactId="EBI-1043398"/>
   <organismsDiffer>false</organismsDiffer>
   <experiments>1</experiments>
</comment>
<comment type="interaction">
   <interactant intactId="EBI-1043398"/>
   <interactant intactId="EBI-1050185">
     <id>Q8NBH6</id>
   </interactant>
   <organismsDiffer>false</organismsDiffer>
   <experiments>1</experiments>
</comment>
<comment type="interaction">
   <interactant intactId="EBI-1043398"/>
   <interactant intactId="EBI-350350">
     <id>P21266</id>
     <label>GSTM3</label>
   </interactant>
   <organismsDiffer>false</organismsDiffer>
   <experiments>1</experiments>
</comment>
```

Related Work

Semantic Framework — *many* related ideas

- [Dayal, Bernstein '82] "exact translation"
- [Bancilhon, Spryatos '81] "constant complement"
- [Gottlob, Paolini, Zicari '88] "dynamic views"
- [Hegner '03] closed vs. open views.

Bijective languages — *many*

Bidirectional languages

- [Meertens] — constaint maintainers; similar laws
- [UTokyo PSD Group] — structured document editors

Lens languages

- [POPL '05, PLAN-X '07] — trees
- [Bohannon et al PODS '06] — relations

See our TOPLAS paper for details...

Extensions and Future work

Primitives:

- composition
- permuting
- filtering

Semantic Foundations:

- quasi-oblivious lenses
- quotient lenses

Optimization:

- algebraic theory
- efficient automata
- streaming lenses

Keys: matching based on similiarity metrics.

Thank You!

Want to play? Boomerang is available for download:

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

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

Extra Slides

Quasi-Obliviousness

We want a property to distinguish the behavior of the first composers lens from the version with chunks and keys.

Intuition: the *put* function is agnostic to the order of chunks having different keys.

Let $\sim \subseteq S \times S$ be the equivalence relation that identifies sources up to key-respecting reorderings of chunks.

The dictionary composers lens obeys

$$\frac{s \sim s'}{l.put\ t\ s = l.put\ t\ s'} \qquad \text{(EquivPut)}$$

but the basic lens does not.

Quasi-Obliviousness

More generally we can let $\sim$ be an arbitrary equivalences on S.

The EquivPut law characterizes some important special cases of lenses:

- Every lens is quasi-oblivious wrt the identity relation.
- Bijective lenses are quasi-oblivious wrt the total relation.
- For experts: Recall the PutPut law:

$$put(t_2, put(t_1, s)) = put(t_2, s)$$

which captures the notion of "constant complement" from databases. A lens obeys this law iff each equivalence classes of the coarsest $\sim$ maps via *get* to T.

Copy and Delete

$$cp\ E \in [\![E]\!] \Longleftrightarrow [\![E]\!]$$

$$\begin{array}{lcl} get\ s & = & s \\ put\ t\ s & = & t \\ create\ t & = & t \end{array}$$

$$\frac{[\![E]\!] \neq \emptyset}{del\ E \in [\![E]\!] \Longleftrightarrow \{\epsilon\}}$$

$$\begin{array}{lcl} get\ s & = & \epsilon \\ put\ \epsilon\ s & = & s \\ create\ \epsilon & = & \mathsf{choose}(E) \end{array}$$

Concatenation

$$\frac{S_1 \mathbin{.^!} S_2 \qquad T_1 \mathbin{.^!} T_2 \qquad l_1 \in S_1 \Longleftrightarrow T_1 \qquad l_2 \in S_2 \Longleftrightarrow T_2}{l_1 \cdot l_2 \in S_1 \cdot S_2 \Longleftrightarrow T_1 \cdot T_2}$$

$$\begin{aligned}
\mathit{get}\ (s_1 \cdot s_2) &= (l_1.\mathit{get}\ s_1) \cdot (l_2.\mathit{get}\ s_2) \\
\mathit{put}\ (t_1 \cdot t_2)\ (s_1 \cdot s_2) &= (l_1.\mathit{put}\ t_1\ s_1) \cdot (l_2.\mathit{put}\ t_2\ s_2) \\
\mathit{create}\ (t_1 \cdot t_2) &= (l_1.\mathit{create}\ t_1) \cdot (l_2.\mathit{create}\ t_2)
\end{aligned}$$

$S_1 \mathbin{.^!} S_2$ means "the concatenation of S_1 and S_2 is uniquely splittable"

Kleene-*

$$\frac{l \in S \Longleftrightarrow T \qquad S^{!*} \qquad T^{!*}}{l^* \in S^* \Longleftrightarrow T^*}$$

$$\begin{aligned}
&get\ (s_1 \cdots s_n) &&= (l.get\ s_1) \cdots (l.get\ s_n) \\
&put\ (t_1 \cdots t_n)\ (s_1 \cdots s_m) &&= (l.put\ t_1\ s_1) \cdots (l.put\ t_m\ s_m) \cdot \\
& &&\quad (l.create\ t_{m+1}) \cdots (l.create\ t_n) \\
&create\ (t_1 \cdots t_n) &&= (l.create\ t_1) \cdots (l.create\ t_n)
\end{aligned}$$

Union

$$\frac{S_1 \cap S_2 = \emptyset \qquad I_1 \in S_1 \iff T_1 \qquad I_2 \in S_2 \iff T_2}{I_1 \mid I_2 \in S_1 \cup S_2 \iff T_1 \cup T_2}$$

$$\begin{aligned}
get\ s &= \begin{cases} I_1.get\ s & \text{if } s \in S_1 \\ I_2.get\ s & \text{if } s \in S_2 \end{cases} \\
put\ t\ s &= \begin{cases} I_i.put\ t\ s & \text{if } s \in S_i \wedge t \in T_i \\ I_j.create\ t & \text{if } s \in S_i \wedge t \in T_j \setminus T_i \end{cases} \\
create\ a &= \begin{cases} I_1.create\ t & \text{if } t \in T_1 \\ I_2.create\ t & \text{if } t \in T_2 \setminus T_1 \end{cases}
\end{aligned}$$

The Essential Dictionary Lens

$$\frac{l \in S \overset{R,D}{\Longleftrightarrow} T}{\texttt{<}l\texttt{>} \in S \overset{\{\square\},D'}{\Longleftrightarrow} T}$$

$$
\begin{array}{lcl}
\texttt{<}l\texttt{>}.get\ s & = & l.get\ s \\
\texttt{<}l\texttt{>}.put\ t\ (\square, d) & = & \pi_1(l.put\ t\ (r, d'')), d' \\
 & & \text{where } (r, d''), d' = lookup\ (l.key\ t)\ d \\
\texttt{<}l\texttt{>}.parse\ s & = & \square, \{(l.key\ (l.get\ s)) \mapsto [s]\}
\end{array}
$$