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
Bidirectional Mappings

S \rightarrow T

T \rightarrow S
Bidirectional Mappings

S → T

T

Updated T

update
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...

Diagram:

- **Document**
- **Screen presentation**
- **Updated document**

- XML Editor
- Edit operation on screen
The View Update Problem In Practice

...and in data synchronizers like the Harmony system.
Linguistic Approach
Terminology

lens
Terminology
Terminology

lens
get
create
Terminology

- get
- put
- lens
- get
- put
A lens \( l \) from \( S \) to \( T \) is a triple of functions

\[
\begin{align*}
l \text{.} \text{get} & \in S \rightarrow T \\
l \text{.} \text{put} & \in T \rightarrow S \rightarrow S \\
l \text{.} \text{create} & \in T \rightarrow S
\end{align*}
\]

obeying three “round-tripping” laws:

\[
\begin{align*}
l \text{.} \text{put} (l \text{.} \text{get} s) s &= s & \text{(GETPUT)} \\
l \text{.} \text{get} (l \text{.} \text{put} t s) &= t & \text{(PUTGET)} \\
l \text{.} \text{get} (l \text{.} \text{create} t) &= t & \text{(CREATEGET)}
\end{align*}
\]
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.
Source string:

"Benjamin Britten, 1913–1976, English"

Target string:

"Benjamin Britten, English"
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"
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
@inproceedings{utts07,
  author = {J. Nathan Foster
            and Benjamin C. Pierce
            and Alan Schmitt},
  title = {A Logic Your Typechecker Can Count On: Unordered Tree Types in Practice},
  booktitle = {PLAN-X},
  year = 2007,
  month = jan,
  pages = {80--90},
  jnf = "yes",
  plclub = "yes",
}
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)

```xml
<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] — constraint 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 similarity 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
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

\[
\frac{s \sim s'}{l.\text{put} \ t \ s = l.\text{put} \ t \ s'}
\]

(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 \( E \in [E] \iff [E] \)

\[
\begin{align*}
\text{get } s &= s \\
\text{put } t \ s &= t \\
\text{create } t &= t
\end{align*}
\]

\[
\begin{align*}
\text{del } E \in [E] &\iff \{\epsilon\} \\
\text{get } s &= \epsilon \\
\text{put } \epsilon \ s &= s \\
\text{create } \epsilon &= \text{choose}(E)
\end{align*}
\]
**Concatenation**

\[
\begin{align*}
S_1 &: S_2 & T_1 &: T_2 \\
\text{l}_1 \in S_1 &\iff T_1 & \text{l}_2 \in S_2 &\iff T_2 \\
\text{l}_1 \cdot \text{l}_2 \in S_1 \cdot S_2 &\iff T_1 \cdot T_2 \\
\end{align*}
\]

\[
\begin{align*}
\text{get} \ (s_1 \cdot s_2) &\quad = \quad (\text{l}_1.\text{get} \ s_1) \cdot (\text{l}_2.\text{get} \ s_2) \\
\text{put} \ (t_1 \cdot t_2) \ (s_1 \cdot s_2) &\quad = \quad (\text{l}_1.\text{put} \ t_1 \ s_1) \cdot (\text{l}_2.\text{put} \ t_2 \ s_2) \\
\text{create} \ (t_1 \cdot t_2) &\quad = \quad (\text{l}_1.\text{create} \ t_1) \cdot (\text{l}_2.\text{create} \ t_2) \\
\end{align*}
\]

\[S_1 \cdot ! S_2\] means “the concatenation of \( S_1 \) and \( S_2 \) is uniquely splittable”
Kleene-\(^*\)

\[
\begin{array}{c}
l \in S \iff T \\
S^* \quad T^* \\
l^* \in S^* \iff T^*
\end{array}
\]

get \((s_1 \cdots s_n)\) = \((l.\text{get } s_1) \cdots (l.\text{get } s_n)\)

put \((t_1 \cdots t_n) (s_1 \cdots s_m)\) = \((l.\text{put } t_1 s_1) \cdots (l.\text{put } t_m s_m) \cdot (l.\text{create } t_{m+1}) \cdots (l.\text{create } t_n)\)

create \((t_1 \cdots t_n)\) = \((l.\text{create } t_1) \cdots (l.\text{create } t_n)\)
Union

\[
S_1 \cap S_2 = \emptyset \quad l_1 \in S_1 \iff T_1 \quad l_2 \in S_2 \iff T_2
\]

\[
l_1 | l_2 \in S_1 \cup S_2 \iff T_1 \cup T_2
\]

\[
\text{get } s = \begin{cases} 
  l_1.\text{get } s & \text{if } s \in S_1 \\
  l_2.\text{get } s & \text{if } s \in S_2 
\end{cases}
\]

\[
\text{put } t s = \begin{cases} 
  l_i.\text{put } t s & \text{if } s \in S_i \land t \in T_i \\
  l_j.\text{create } t & \text{if } s \in S_i \land t \in T_j \setminus T_i 
\end{cases}
\]

\[
\text{create } a = \begin{cases} 
  l_1.\text{create } t & \text{if } t \in T_1 \\
  l_2.\text{create } t & \text{if } t \in T_2 \setminus T_1
\end{cases}
\]
The Essential Dictionary Lens

\[
\frac{l \in S \leftrightarrow^{R,D} T}{\langle l \rangle \in S \leftrightarrow^{\{\Box\},D'} T}
\]

\[
\begin{align*}
\langle l \rangle \text{.get } s &= l \text{.get } s \\
\langle l \rangle \text{.put } t (\Box, d) &= \pi_1(l \text{.put } t (r, d'')), d' \\
\text{where } (r, d''), d' &= \text{lookup } (l \text{.key } t) \; d \\
\langle l \rangle \text{.parse } s &= \Box, \{(l \text{.key } (l \text{.get } s)) \mapsto [s]\}
\end{align*}
\]