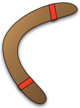
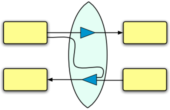
# Bidirectional Programming Languages

Nate Foster
University of Pennsylvania

April 2009






redact
exchange
replicate
convert
integrate
query
analyze
update
mashup
transform
clean
curate
reconcile
data
modify
hide
maintain
synchronize
extract
summarize
evolve

We *can* write complicated data transformations in C...


or Java...


or C++...

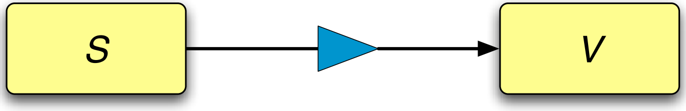
...or a tool specifically designed for the task!

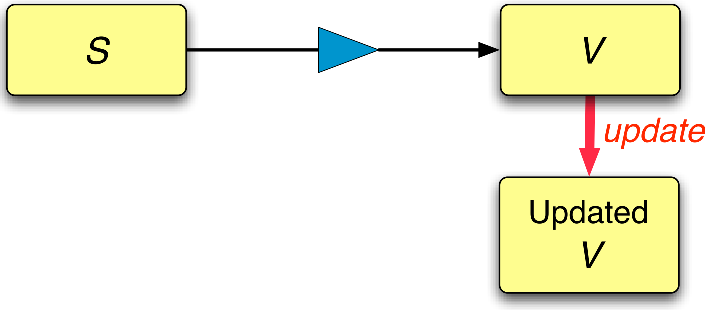
# Domain-specific languages

- Clean semantics
- Natural syntax
- Better tools

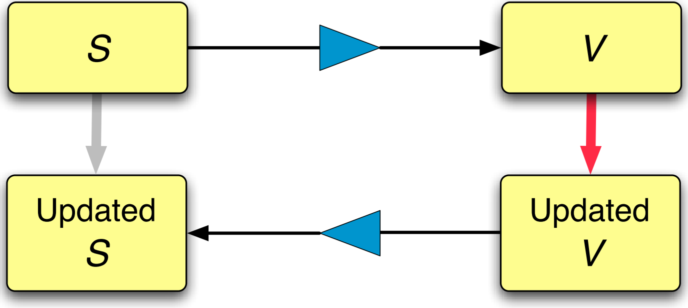

data
update
query
mashup
transform
hide
maintain
replicate
integrate
redact
convert
exchange
clean
modify
synchronize
reconcile
extract
summarize
evolve
curate
analyze


redact
convert
exchange
replicate
analyze
update
integrate
query
mashup
transform
clean
curate
reconcile
data
modify
hide
maintain
synchronize
extract
summarize
evolve


S
V

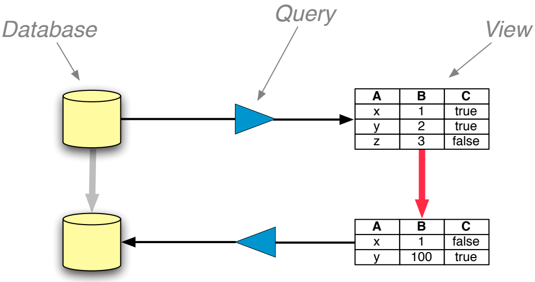


S
V
update
Updated
V


S
V
Updated S
Updated V

# The View Update Problem

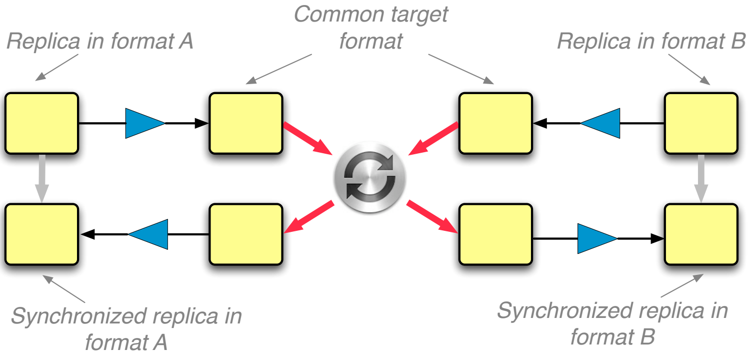
In databases, this is known as the view update problem.




[Bancilhon, Spryatos '81]

# The View Update Problem In Practice

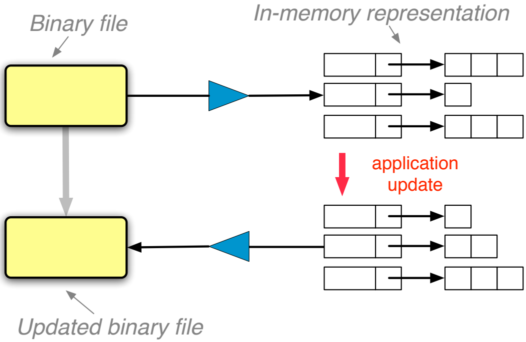
It also arises in data converters and synchronizers...




[Foster, Greenwald, Pierce, Schmitt JCSS '07]— Harmony

# The View Update Problem In Practice

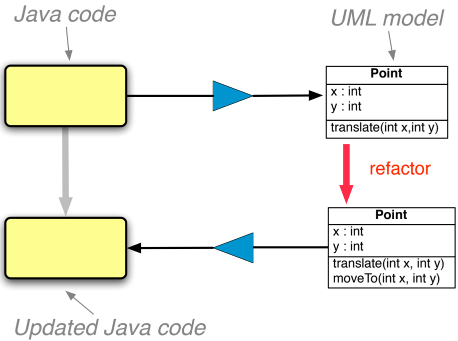
...in picklers and unpicklers...




[Fisher, Gruber '05]— PADS

# The View Update Problem In Practice

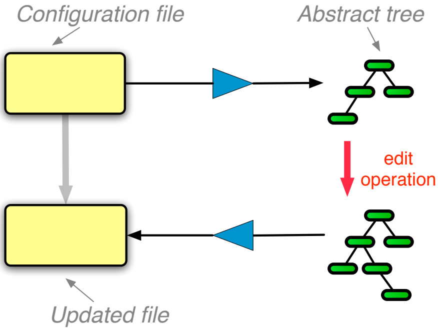
...in model-driven software development...




[Stevens '07]— bidirectional model transformations

# The View Update Problem In Practice

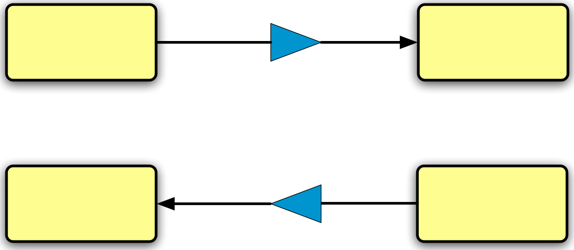
...in tools for managing operating system configurations...




[Lutterkort '08]— Augeas

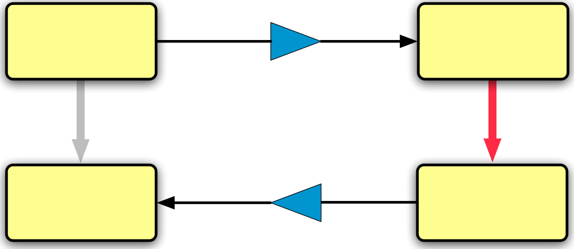
# Problem

How do we write these bidirectional transformations?



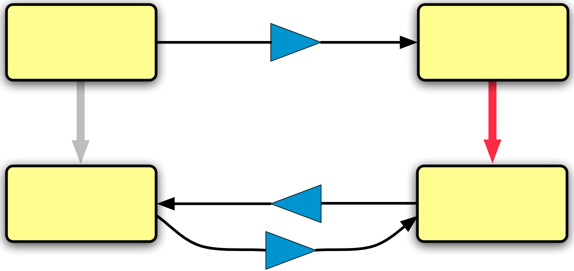
# Problem: Why is it hard?

We want updates to the view to be translated "exactly"...



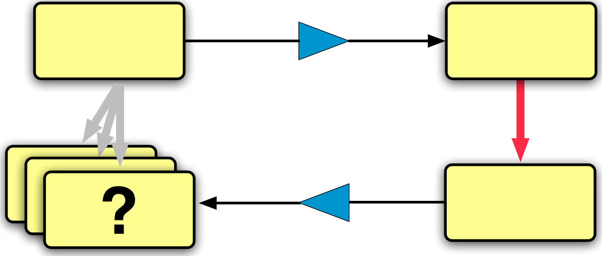
# Problem: Why is it hard?

We want updates to the view to be translated “exactly”...



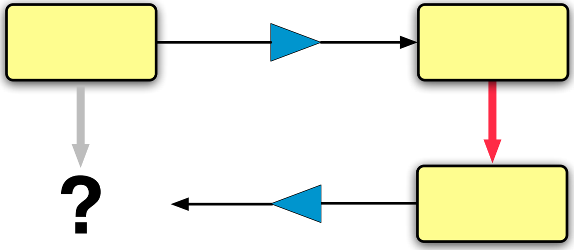
# Problem: Why is it hard?

...but some updates have *many* corresponding source updates...

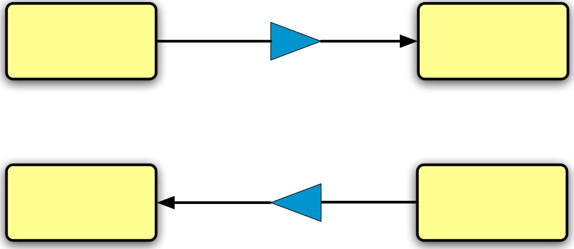


# Problem: Why is it hard?

...while others have *none*!



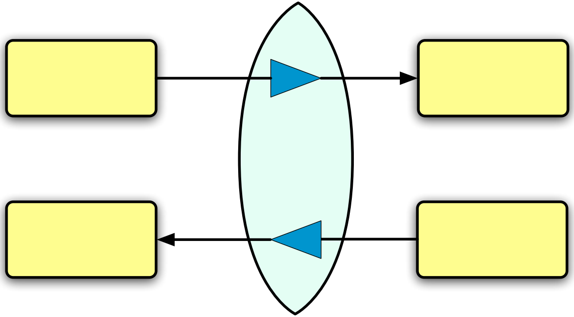
# Possible Approaches



**Bad:** write the two transformations as separate functions.

- tedious to program
- difficult to get right
- a nightmare to maintain

# Possible Approaches



**Good:** derive both transformations from the same program.

- Clean semantics: behavioral laws guide language design
- Natural syntax: parsimonious and compositional
- Better tools: type system guarantees well-behavedness

# This talk: Goal

"Bidirectional programming languages are an effective and elegant means of describing updatable views"

# This talk: Outline

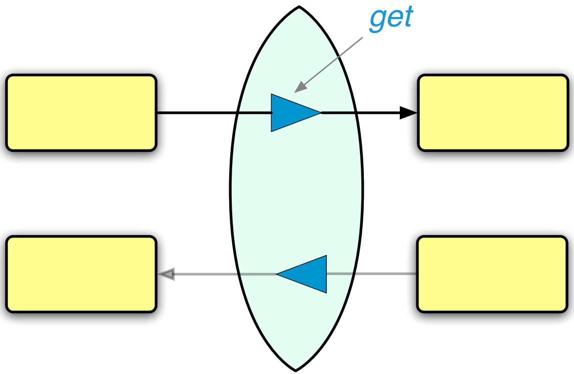
1. Lenses
   - Design goals
   - Semantics
2. String Lenses
   - Core operators
   - Type system
3. Boomerang
   - Ordered data
   - Ignorable data
   - Implementation & Applications
4. Ongoing Work
   - Updatable Security Views
5. Future Directions
   - Data provenance
   - Model transformations

[Foster, Greenwald, Moore, Pierce, Schmitt TOPLAS '07]

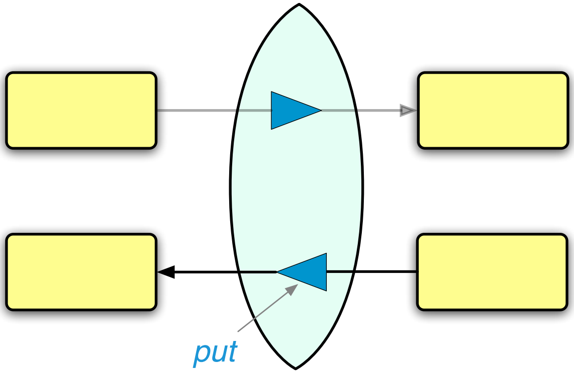
# Lenses

''‘Never look back unless
you are planning to go that way"
—H D Thoreau

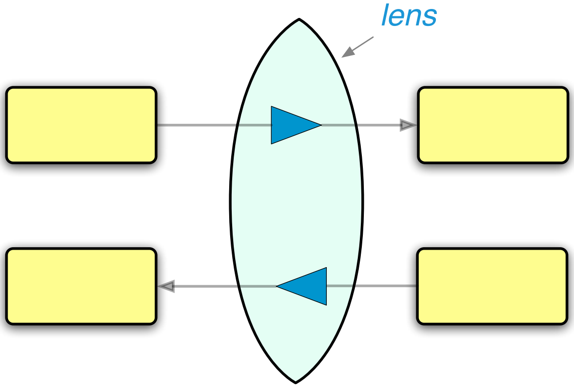
# Terminology



# Terminology



# Terminology



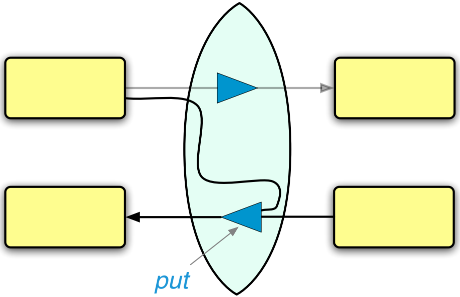
# Bidirectional vs. Bijective

Goal #1: lenses should be capable of hiding source data.

# Bidirectional vs. Bijective

Goal #1: lenses should be capable of hiding source data.

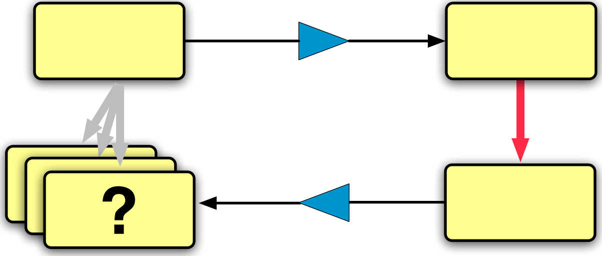
- In general, **get** may be non-injective
- and so **put** needs to take the original source as an argument




(Of course, the purely bijective case is also very interesting.)

# Choice of Put Function

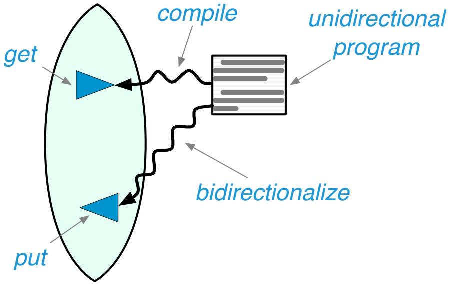
Recall that for some view updates there are *many* corresponding source updates.



# Choice of Put Function

Goal #2: programmers should be able to choose a **put** function that embodies an appropriate policy for propagating updates back to sources.

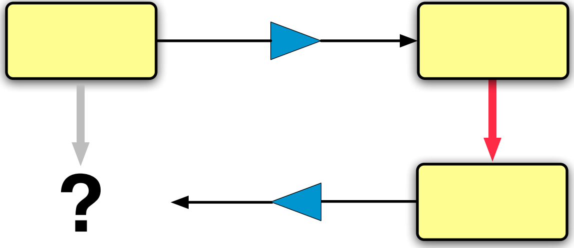
“Bidirectionalization” appears attractive...




...but does not provide a way to make this choice.

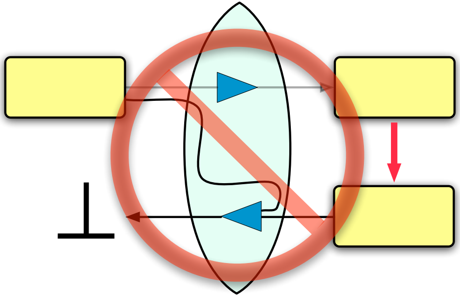
# Totality

Recall that some view updates do not have *any* corresponding source updates.



# Totality

Goal #3: the **put** function should be a total function, capable of doing *something* reasonable with every view and source.




Totality ensures that the view is a robust abstraction, but forces us to use an extremely precise type system.

# Well-Behaved Lenses

A lens $l$ mapping between a set $S$ of sources and $V$ of view is a pair of total functions

$$\begin{aligned} l.\mathbf{get} &\in S \rightarrow V \\ l.\mathbf{put} &\in V \rightarrow S \rightarrow S \end{aligned}$$

obeying "round-tripping" laws

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

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

for every $s \in S$ and $v \in V$.

# Related Frameworks

Databases: *many* related ideas

- [Dayal, Bernstein '82] "exact translation"
- [Bancilhon, Spryatos '81] "constant complement"
- [Gottlob, Paolini, Zicari '88] "dynamic views"

User Interfaces: [Meertens '98] "constraint maintainers"

See [Foster *et. al* TOPLAS '07] for details...

# Related Languages

Harmony Group @ Penn

- [Foster *et al.* TOPLAS '07] — trees
- [Bohannon, Pierce, Vaughan PODS '06] — relations
- [Foster *et al.* JCSS '07] — data synchronizer

Bijective languages

- [PADS Project @ AT&T] — picklers and unpicklers
- [Hosoya, Kawanaka '06] — biXid
- [Braband, Møller, Schwartzbach '05] — XSugar

Bidirectional languages

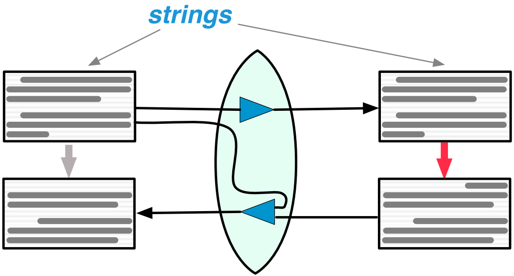
- [PSD @ Tokyo] — "bidirectionalization", structure editors
- [Gibbons, Wang @ Oxford] — Wadler's views
- [Voïgtlaender '09] — bidirectionalization "for free"
- [Stevens '07] — lenses for model transformations

[Bohannon, Foster, Pierce, Pilkiewicz, Schmitt POPL '08]

# String Lenses

"The art of progress is
to preserve order amid change
and to preserve change amid order."
—A N Whitehead

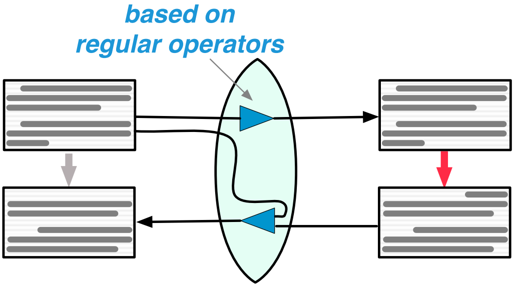
# Data Model




Why strings?

1. Simple setting $\longrightarrow$ exposes fundamental issues
2. There's a lot of string data in the world
3. Programmers are already comfortable with regular operators (union, concatenation, and Kleene star)

# Computation Model

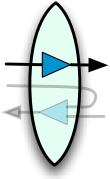



Why strings?

1. Simple setting $\longrightarrow$ exposes fundamental issues
2. There's a lot of string data in the world
3. Programmers are already comfortable with regular operators (union, concatenation, and Kleene star)

# Example: Redacting Lens (Get)

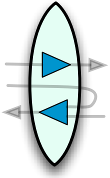
```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```

# Example: Redacting Lens (Update)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```



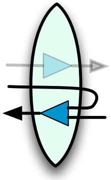
```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```

```
08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
```

# Example: Redacting Lens (Put)

```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClu (Seminar room)
*15:00 Workout (Gym)
```

```
08:30 BUSY
12:15 PLClu
15:00 BUSY
```



```
*08:30 Coffee with Sara (Starbucks)
 12:15 PLClub (Seminar room)
*15:00 Workout (Gym)
 16:00 Meeting (Unknown)
```

```
08:30 BUSY
12:15 PLClub
15:00 BUSY
16:00 Meeting
```

# Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

# Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN
(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL
(* main lens *)
let redact : lens = event*
```

# Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

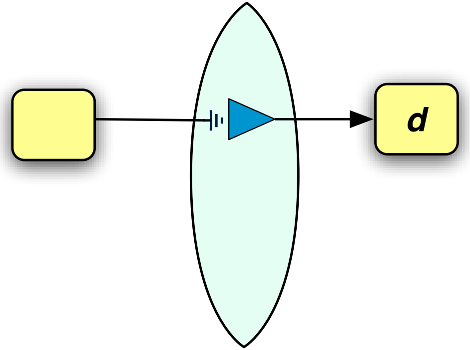
# Example: Redacting Lens (Definition)

```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

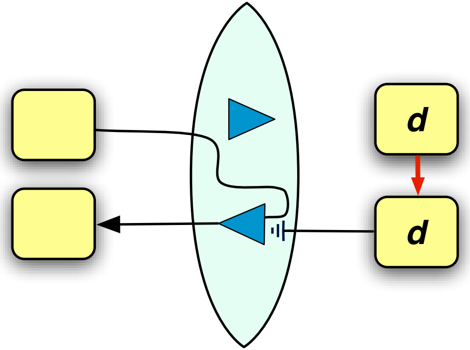
# Example: Redacting Lens (Definition)

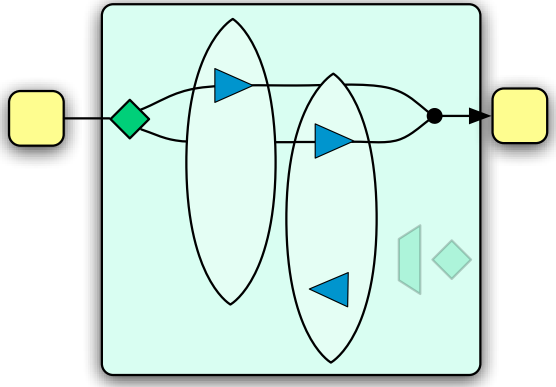
```
(* regular expressions *)
let TEXT : regexp = ([^\n\\()] | "\\(" | "\\)" | "\\\\")*
let TIME : regexp = DIGIT{2} . COLON . DIGIT{2} . SPACE
let LOCATION : regexp = SPACE . LPAREN . TEXT . RPAREN

(* helper lenses *)
let public : lens =
  del SPACE .
  copy TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  copy TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)"

let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = event*
```

## $E \leftrightarrow d$ (Get)

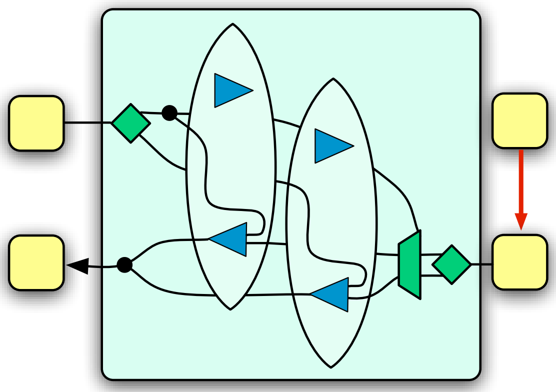


# $E \leftrightarrow d$ (Put)



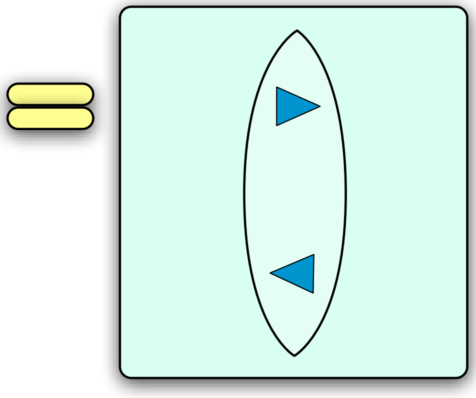


Type system ensures that choice is deterministic.

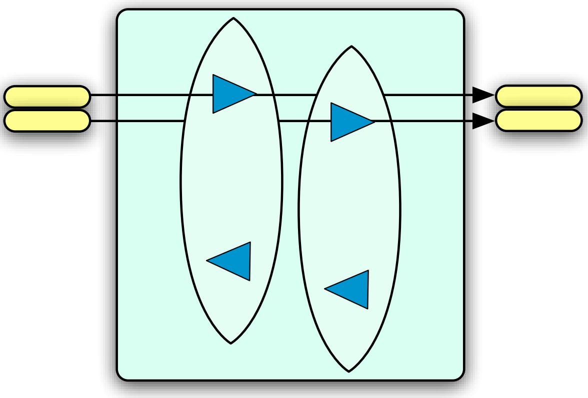


Type system ensures that choice is deterministic.

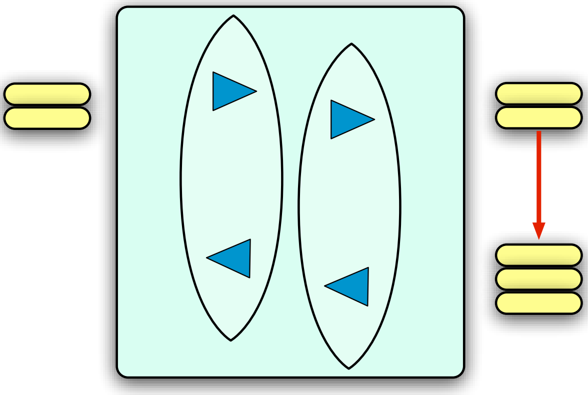
# l*            (Get)



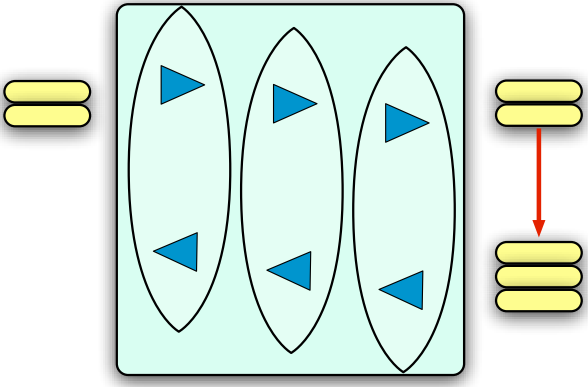
# l* (Get)



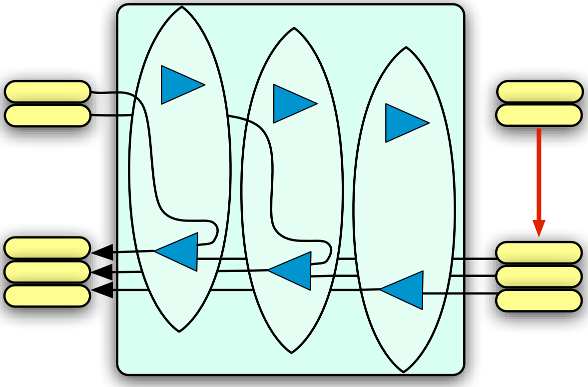
# I* (Put)



# /*                (Put)



# /* (Put)



Type system ensures that strings are split the same way.

# String Lens Type System

Based on regular expression types...

# String Lens Type System

Based on regular expression types...

$$\frac{}{\mathit{copy}\ E \in [\![E]\!] \Longleftrightarrow [\![E]\!]} \qquad \frac{}{E \leftrightarrow d \in [\![E]\!] \Longleftrightarrow \{d\}}$$

$$\frac{l \in S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\mathit{default}\ l\ d \in S \Longleftrightarrow V} \qquad \frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cdot^! S_2 \\ l_2 \in S_2 \Longleftrightarrow V_2 & V_1 \cdot^! V_2 \end{array}}{(l_1 \cdot l_2) \in S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cap S_2 = \emptyset \\ l_2 \in S_2 \Longleftrightarrow V_2 & \end{array}}{(l_1 \mid l_2) \in S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2} \qquad \frac{l \in S \Longleftrightarrow V \qquad S^{!*} \qquad V^{!*}}{l^* \in S^* \Longleftrightarrow V^*}$$

$S_1 \cdot^! S_2$ (or $S^{!*}$) means that the concatenation (or iteration) is unambiguous.

# String Lens Type System

Based on regular expression types...

$$\frac{}{\textit{copy } E \in [\![E]\!] \Longleftrightarrow [\![E]\!]}$$

$$\frac{}{E \leftrightarrow d \in [\![E]\!] \Longleftrightarrow \{d\}}$$

$$\frac{l \in S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\textit{default } l \ d \in S \Longleftrightarrow V}$$

$$\frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cdot^! S_2 \\ l_2 \in S_2 \Longleftrightarrow V_2 & V_1 \cdot^! V_2 \end{array}}{(l_1 \cdot l_2) \in S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cap S_2 = \emptyset \\ l_2 \in S_2 \Longleftrightarrow V_2 & \end{array}}{(l_1 \mid l_2) \in S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2}$$

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

$S_1 \cdot^! S_2$ (or $S^{!*}$) means that the concatenation (or iteration) is unambiguous.

# String Lens Type System

Based on regular expression types...

$$\frac{}{\textit{copy}\ E \in [\![E]\!] \Longleftrightarrow [\![E]\!]} \qquad \frac{}{E \leftrightarrow d \in [\![E]\!] \Longleftrightarrow \{d\}}$$

$$\frac{l \in S \Longleftrightarrow V \qquad d \in [\![S]\!]}{\textit{default}\ l\ d \in S \Longleftrightarrow V} \qquad \frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cdot^! S_2 \\ l_2 \in S_2 \Longleftrightarrow V_2 & V_1 \cdot^! V_2 \end{array}}{(l_1 \cdot l_2) \in S_1 \cdot S_2 \Longleftrightarrow V_1 \cdot V_2}$$

$$\frac{\begin{array}{ll} l_1 \in S_1 \Longleftrightarrow V_1 & S_1 \cap S_2 = \emptyset \\ l_2 \in S_2 \Longleftrightarrow V_2 & \end{array}}{(l_1 \mid l_2) \in S_1 \cup S_2 \Longleftrightarrow V_1 \cup V_2} \qquad \frac{l \in S \Longleftrightarrow V \qquad S^{!*} \qquad V^{!*}}{l^* \in S^* \Longleftrightarrow V^*}$$

$S_1 \cdot^! S_2$ (or $S^{!*}$) means that the concatenation (or iteration) is unambiguous.

**Theorem**

If $l \in S \Longleftrightarrow V$ then $l$ is a well-behaved lens.

# Comparison: Separate Functions

```
module B = Buffer
module R = Str
module L = List
module U = Unix
let rx = R.regexp
let sch = R.search_forward

(* helpers *)
let error s =
  print_string s;
  exit 1

let exec s =
  let buf = B.create 17 in
  let p = U.open_process_in s in
  (try
    while true do B.add_char buf (input_char p) done
  with End_of_file -> ());
  ignore (U.close_process_in p);
  B.contents buf

(* regexps *)
let begin_event = rx "BEGIN:VEVENT"
let text = "[^\n]*"
let esctext = "([^\n\\()]\\|\\\\\\|\\(\\|\\))*"
let date = "[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]T[0-9][0-9][0-9][0-9][0-9][0-9]Z"
let codes = [("\\", "\\\\"); ("(","\\("); (")","\\)")]

let today () = exec "date +%Y%m%d | tr -d '\n'"
let now () = exec "date +%H%M%S | tr -d '\n'"
let uid () = exec "cat /dev/urandom|uuencode -m -|tail +2|tr -dc 'a-zA-Z0-9'|head -c25"

(* dictionary lens helpers *)
let escape codes s =
  L.fold_right
    (fun (p,r) s -> R.global_replace (rx p) r s)
    codes
    s

let unescape codes s =
  L.fold_left
    (fun s (p,r) -> R.global_replace (rx r) p s)
    s codes

let distance s1 s2 =
  let m,n = String.length s1, String.length s2 in
  let aodd,aeven = Array.make (succ n) 0,Array.make (succ n) 0 in
    for j=0 to n do aeven.(j) <- j done;
    for i=1 to m do
      let apredi = if i mod 2 = 0 then aodd else aeven in
      let ai = if i mod 2 = 0 then aeven else aodd in
        ai.(0) <- i;
        for j = 1 to n do
          let x =
            if String.get s1 (pred i) = String.get s2 (pred j) then 0 else 1 in
          ai.(j) <-
            (min (apredi.(j) + 1) (min (ai.(j-1) + 1) (apredi.(j-1) + x)));
        done
    done;
    if m mod 2 = 0 then aeven.(n) else aodd.(n)

let rec lookup k l =
  let l' = L.map (fun (ki,ci) -> (distance ki k,ki,ci)) l in
  let rec aux acc l = match acc,l with
      None,[] -> None
      Some(_,ki,ci),[] -> Some(ki,ci)
      Some(di,ki,ci),(dj,kj,cj)::rest when dj < di -> aux (Some(dj,kj,cj)) rest
      None,(dj,kj,cj)::rest -> aux (Some(dj,kj,cj)) rest
      _,_::rest -> aux acc rest in
  aux None l'

let remove k l =
  let rec loop acc = function
      [] -> L.rev_append acc []
      h::t when k = fst h -> L.rev_append acc t
      h::t -> loop (h::acc) t in
  loop [] l

let wrap b =
  let b_len = String.length b in
  let buf,line_buf,aux_buf = B.create b_len, B.create 75, B.create 75 in
  let do_it s b =
    if B.length line_buf <> 0 then B.add_string buf s;
    B.add_buffer buf b;
    B.reset b in
  let rec loop i =
    let sum =
      let aux_len = B.length aux_buf in
      let line_len = let n = B.length line_buf in if n=0 then n else succ n in
      aux_len + line_len in
    if sum > 75 then do_it "\n" line_buf;
```

Helpers

```
    if i = b_len then (do_it " " aux_buf; do_it "\n" line_buf)
    else
      (if b.[i] = ' ' then do_it " " aux_buf
       else B.add_char aux_buf b.[i];
       loop (succ i)) in
  loop 0;
  B.contents buf

let unwrap c = R.global_replace (rx "\n") (String.make 1 ' ') c

let field tag r s =
  try
    ignore(sch (rx (tag ^ ":" ^ "\\(" ^ r ^ "\\)\n")) s 0);
    R.matched_group 1 s
  with Not_found -> error ("Couldn't find " ^ tag ^ " in " ^ s ^ "\n")

let times s = String.sub s 9 2, String.sub s 11 2
```

```
let get c =
  let buf = B.create 17 in
  let add = B.add_string buf in
  let events = L.tl (R.split begin_event c) in
  let event e =
    let p = field "CLASS" "PUBLIC\\|PRIVATE" e in
    let t1,t2 = times (field "DTSTART" date e) in
    let s = escape codes (field "SUMMARY" text e) in
    let l = escape codes (field "LOCATION" text e) in
    if p = "PUBLIC" then add " " else add "*";
    add (t1 ^ ":" ^ t2 ^ " ");
    add (wrap s ^ " (" ^ l ^ ")");
    add "\n" in
  L.iter event events;
  B.contents buf
```

Source to View

```
let put a c =
  let is_noop a c = a ^ "\n" = get ("BEGIN:VCALENDAR\nBEGIN:VEVENT" ^ c ^ "END:VCALENDAR\n") in
  let buf = B.create 17 in
  let line s v = B.add_string buf (s ^ ":" ^ wrap v ^ "\n") in
  let c_line s v c = B.add_string buf (s ^ ":" ^ field s v c ^ "\n") in
  let preamble,c_events = match R.split begin_event c with
      [] -> error "ill-formed"
      h::t -> (h,t) in
  let c_assoc = L.map (fun c -> (field "SUMMARY" text c ^ field "LOCATION" text c,c)) c_events in
  let a_events =
    L.map
      (fun a ->
         let p = a.[0] = ' ' in
         let t = (String.sub a 1 2) ^ (String.sub a 4 2) in
         ignore (sch (rx ("[ *][0-9][0-9]:[0-9][0-9] \\(" ^ esctext ^ "\\) (\\(" ^ esctext ^ "\\))")) a 0);
         (p,t,R.matched_group 1 a,R.matched_group 2 a,a))
      (R.split (rx "\n") (unwrap a)) in
  let event dict (p,t,s,l,a) =
    line "BEGIN" "EVENT";
    if p then line "CLASS" "PUBLIC"
    else line "CLASS" "PRIVATE";
    let dict' = match lookup (s ^ l) dict with
        Some (k,c) ->
          let dtstart = field "DTSTART" date c in
          let ymd,s = String.sub dtstart 0 8,String.sub dtstart 13 2 in
          line "DTSTART" (ymd ^ "T" ^ t ^ s ^ "Z");
          c_line "DTEND" date c;
          c_line "DTSTAMP" date c;
          c_line "CREATED" date c;
          if is_noop a c then c_line "LAST-MODIFIED" date c
          else line "LAST-MODIFIED" (today () ^ "T" ^ now () ^ "Z");
          c_line "UID" text c;
          c_line "TRANSP" text c;
          c_line "DESCRIPTION" text c;
          c_line "STATUS" text c;
          c_line "SEQUENCE" text c;
          (remove k dict)
        None ->
          line "DTSTART" (today () ^ "T" ^ t ^ "00Z");
          line "DTEND" (today () ^ "T" ^ t ^ "00Z");
          line "DTSTAMP" (today () ^ "T" ^ now () ^ "Z");
          line "CREATED" (today () ^ "T" ^ now () ^ "Z");
          line "LAST-MODIFIED" (today () ^ "T" ^ now () ^ "Z");
          line "UID" (uid ());
          line "TRANSP" "OPAQUE";
          line "DESCRIPTION" "";
          line "STATUS" "TENTATIVE";
          line "SEQUENCE" "0";
          dict in
    line "SUMMARY" (unescape codes s);
    line "LOCATION" (unescape codes l);
    line "END" "EVENT";
    dict' in
  B.add_string buf preamble;
  let _ = L.fold_left event c_assoc a_events in
  B.add_string buf "END:VCALENDAR\n";
  B.contents buf
```

View to Source

# Comparison: String Lens

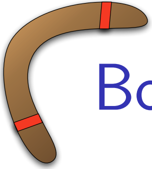
## Helpers

```
let BEGIN_CALENDAR,END_CALENDAR : (regexp * regexp) = "BEGIN:VCALENDAR","END:VCALENDAR"
let BEGIN_EVENT,END_EVENT : (regexp * regexp) = "BEGIN:VEVENT","END:VEVENT"
let PREAMBLE,POSTAMBLE : (regexp * regexp) = (BEGIN_CALENDAR . NL . ( ANY - containing BEGIN_EVENT)),(END_CALENDAR . NL)
let text : lens = Escaping.escape [^\n] #char*string[('\',"\\\\");('(',"\\(");(')',"\\)")]
let TEXT,ESCTEXT : regexp * regexp = ctype text,atype text
let TIME,LOC : regexp * regexp = (DIGIT2 . ":" . DIGIT2), (SPACE . LPAREN . ESCTEXT . RPAREN)

let line (tag:string) (l:lens) : lens = del (tag . ":") . l . del "\n"
let parens (l:lens) : lens = ins "(" . l . ins ")"
let wrapl (l:lens) = left_quot (columnize 75 (ctype l [illegible]
let wrapr (l:lens) = right_quot l (columnize 75 (atype l) [illegible]
let now (s:string) : string = exec "date +%H%M%S | tr -d '\n'"
let today (s:string) : string = exec "date +%Y%m%d | tr -d '\n'"
let uid (s:string) : string = exec "cat /dev/urandom|uuencode -m -|tail +2|tr -dc 'a-zA-Z0-9'|head -c25"
let date (tag:string) (time:regexp -> regexp -> regexp -> lens) : lens =
  line tag (clobber DIGIT8 "" today . del "T" . time DIGIT2 DIGIT2 DIGIT2 . del "Z")
let copy_time (H:regexp) (M:regexp) (S:regexp) : lens = copy H . ins ":" . copy M . del S
let clobber_time (H:regexp) (M:regexp) (S:regexp) : lens = clobber (H . M . S) "" now
```

## Source to View *and* View to Source

```
let event1 : lens =
  del (BEGIN_EVENT . NL) .
  fiat (Sort.sort_concat
    #lens[
      line "CLASS" ("PRIVATE" <-> ASTERISK | "PUBLIC" <-> SPACE)
    ; date "DTSTART" copy_time . ins DASH
    ; date "DTEND" copy_time . ins SPACE
    ; date "DTSTAMP" clobber_time
    ; date "CREATED" clobber_time
    ; date "LAST-MODIFIED" clobber_time
    ; line "UID" (clobber TEXT "" uid)
    ; line "TRANSP" (const TEXT "" "OPAQUE")
    ; line "DESCRIPTION" (del TEXT)
    ; line "STATUS" (const TEXT "" "TENTATIVE")
    ; line "SEQUENCE" (const NUMBER "" "0")
    ; line "SUMMARY" (wrapl (text; key ESCTEXT) . ins SPACE
    ; line "LOCATION" (parens (wrapl (text; key ESCTEXT))) ]) .
  del (END_EVENT) . copy NL
let event2 : lens =
  copy (SPACE | ASTERISK) . merge_with_sep TIME DASH .
  copy SPACE . wrapr (key ESCTEXT . copy LOC) . copy NL
let ical1 : lens = del PREAMBLE . <~ e1:event1 >* . del POSTAMBLE
let ical : lens = ical1; <~ e2:event2 >*
```

[Bohannon, Foster, Pierce, Pilkiewicz, Schmitt POPL '08]
[Foster, Pierce, Pilkiewicz ICFP '08]



# Boomerang

"Good men must not obey
the laws too well"
—R W Emerson

# Challenge: Ignorable Data

Many real-world data formats contain inessential data.

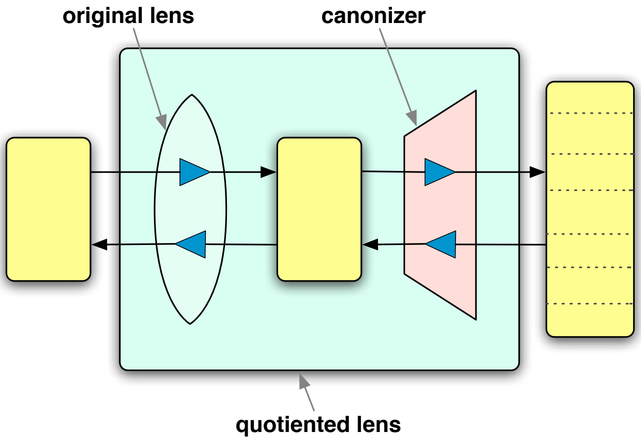
- whitespace, wrapping of long lines of text
- order of fields in record-structured data
- escaping of special characters
- aggregate values, timestamps, etc.

In practice, to handle these details, we need lenses that are well behaved modulo equivalence relations on the source and view.

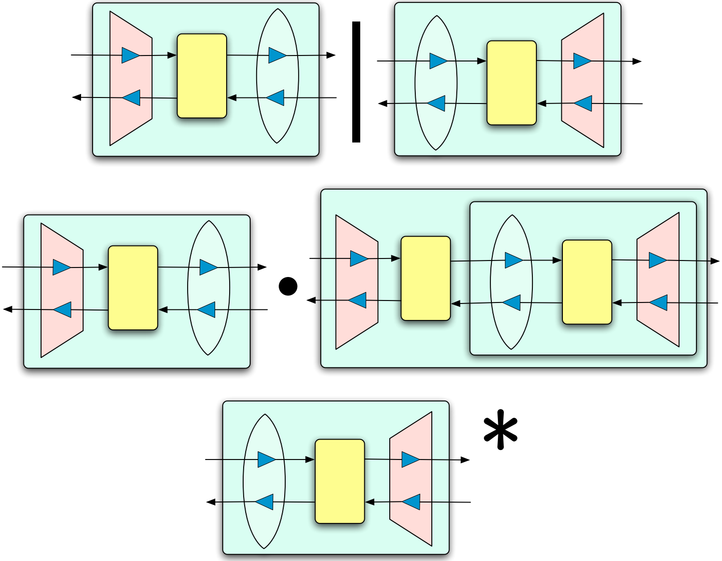
$$l.\mathbf{get}\ (l.\mathbf{put}\ v\ s) \sim_V v \qquad (\textsc{PutGet})$$

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

# Quotient Lenses



# Quotient Lenses



# Challenge: Ordered Data

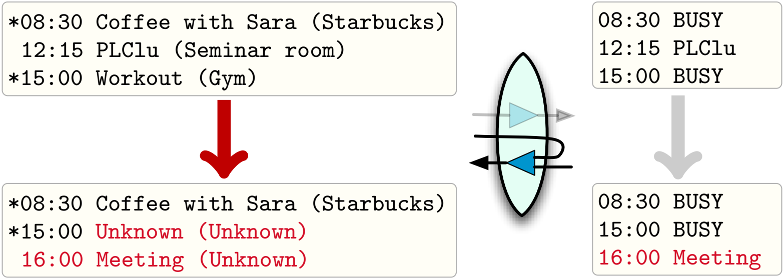
The lenses we have seen so far align data by position.

But, in practice, we often need to align data according to different criteria—e.g., by key.

# Challenge: Ordered Data

The lenses we have seen so far align data by position.

But, in practice, we often need to align data according to different criteria—e.g., by key.



# A Better Redact Lens

Similar to previous version but with a key annotations and a combinator (<l>) that identifies "chunks"

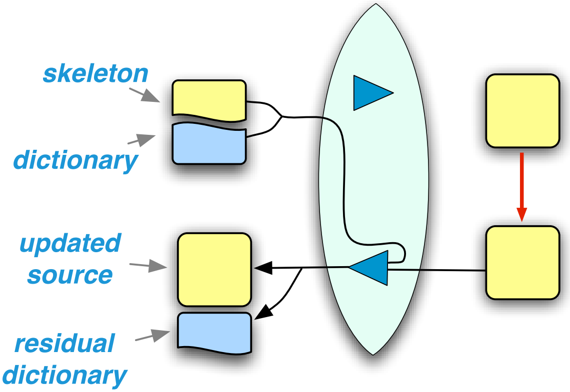
```
(* helper lenses *)
let location : lens = default (del LOCATION) " (Unknown)"

let public : lens =
  del SPACE .
  key TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  key TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)" .
let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = <~ event>*
```

# A Better Redact Lens

Similar to previous version but with a key annotations and a combinator (<l>) that identifies "chunks"

```
(* helper lenses *)
let location : lens = default (del LOCATION) " (Unknown)"

let public : lens =
  del SPACE .
  key TIME .
  copy TEXT .
  default (del LOCATION) " (Unknown)"

let private : lens =
  del ASTERISK .
  key TIME .
  default (TEXT . LOCATION <-> "BUSY") "Unknown (Unknown)" .
let event : lens =
  (public | private) .
  copy NL

(* main lens *)
let redact : lens = <~ event>*
```

# Dictionary Lenses

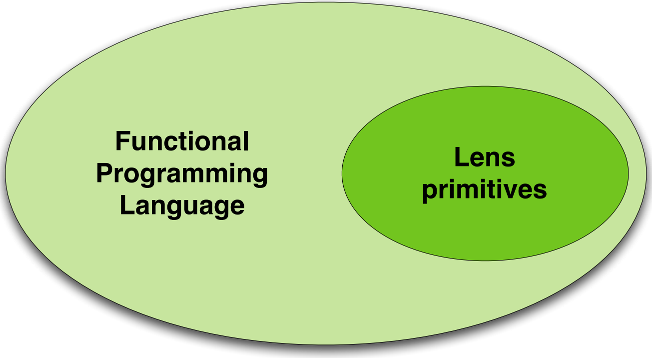



The **put** function works on a dictionary structure where chunks are organized by key.

# Challenge: Language Design

Writing big programs only using combinators would not be fun!

Boomerang is a full-blown functional language over the base types `string`, `regexp`, `lens`, ...



# Additional Features

Boomerang has *many* other lens primitives

- partition
- filter
- permute
- sort
- duplicate
- merge
- sequentially compose
- columnize
- normalize
- clobber
- probe
- etc.

and an extremely rich type system

- regular expression types
- dependent types
- refinement types
- polymorphism
- user-defined datatypes
- modules

implemented in hybrid style [Flanagan '06][Findler, Wadler '09]

# Challenge: Typechecker Engineering

Typechecking uses *many* automata-theoretic operations.

- "Expensive" operations like intersection, difference, and interleaving are used often in practice
- Algorithms for checking ambiguity are computationally expensive rarely implemented

Implementation strategy:

- Compile compact automata [Brzozoswki '64]
- Aggresive memoization [Foster *et al.* PLAN-X '07]

# The Boomerang System

## Lenses

- Bibliographies (BibTeX, RIS)
- Address Books (vCard, XML, ASCII)
- Calendars (iCal, XML, ASCII)
- Scientific Data (SwissProt, UniProtKB)
- Documents (MediaWiki, literate source code)
- Apple Preference Lists (e.g., iTunes)
- CSV

## Libraries

- Escaping
- Sorting
- Lists
- XML

## System

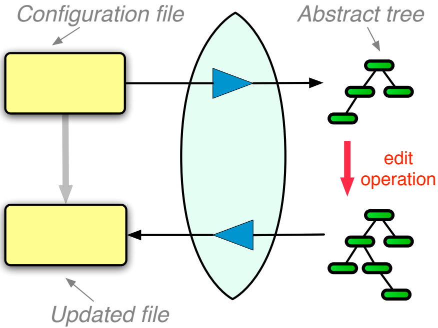
- Stable prototype complete
- Available under LGPL

## Unison Integration

- On the way...

# Boomerang in Industry

"a configuration API."



# Boomerang in Industry

Augeas "a configuration API."

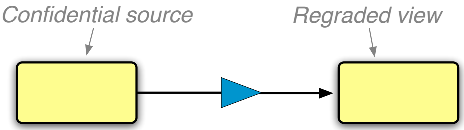
| | | | | |
|---|---|---|---|---|
| aliases.aug | fstab.aug | monit.aug | rsyncd.aug | sysctl.aug |
| aptpreferences.aug | gdm.aug | ntp.aug | samba.aug | util.aug |
| aptsources.aug | group.aug | openvpn.aug | services.aug | vsftpd.aug |
| bbhosts.aug | grub.aug | pam.aug | shellvars.aug | webmin.aug |
| darkice.aug | hosts.aug | passwd.aug | slapd.aug | xinetd.aug |
| dhclient.aug | inifile.aug | php.aug | soma.aug | xorg.aug |
| dnsmasq.aug | inittab.aug | phpvars.aug | spacevars.aug | yum.aug |
| dpkg.aug | interfaces.aug | postfix_main.aug | squid.aug | |
| dput.aug | limits.aug | postfix_master.aug | sshd.aug | |
| exports.aug | logrotate.aug | puppet.aug | sudoers.aug | |

Also used in

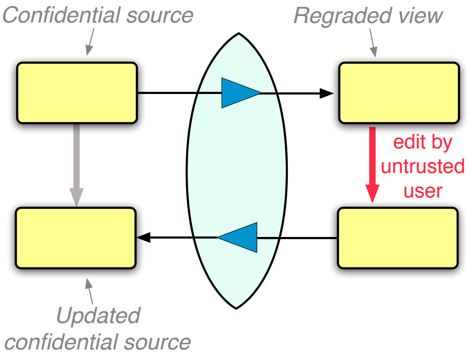
- Puppet – declarative configuration management tool
- Show – SQL-like queries on the filesystem
- Netcf – a network configuration library

# Ongoing Work

# Security Views

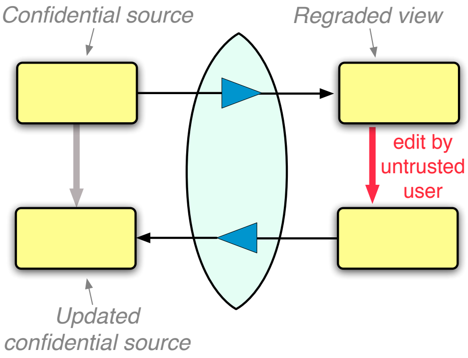


# Updatable Security Views




[Foster, Pierce, Zdancewic CSF '09]

# Requirements for Updatable Security Views

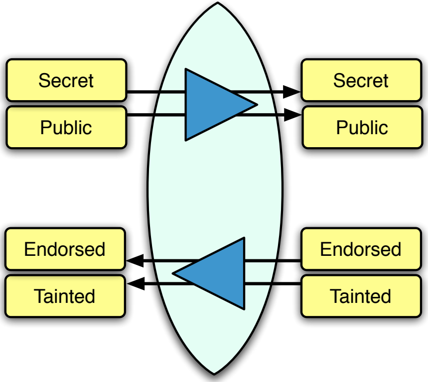



[Foster, Pierce, Zdancewic CSF '09]

1. Confidentiality: **get** does not leak secret data
2. Integrity: **put** does not taint endorsed data

# Non-interference

Requirements can be formulated as non-interference properties.



# Secure Lenses

To distinguish high and low-security data we use equivalences

- $\sim_k$ — "agree on $k$-public data"
- $\approx_k$ — "agree on $k$-endorsed data"

# Secure Lenses

To distinguish high and low-security data we use equivalences

- $\sim_k$ — "agree on $k$-public data"
- $\approx_k$ — "agree on $k$-endorsed data"

described using annotated regular expressions.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}{\cdot}\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}^* \mid \mathcal{R}:k$$

# Secure Lenses

To distinguish high and low-security data we use equivalences

- $\sim_k$ — "agree on $k$-public data"
- $\approx_k$ — "agree on $k$-endorsed data"

described using annotated regular expressions.

$$\mathcal{R} ::= \emptyset \mid u \mid \mathcal{R}{\cdot}\mathcal{R} \mid \mathcal{R}|\mathcal{R} \mid \mathcal{R}* \mid \mathcal{R}:k$$

A secure lens obeys refined laws:

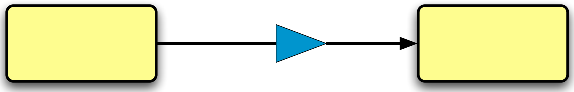
$$\frac{s \sim_k s'}{l.\mathbf{get}\ s \sim_k l.\mathbf{get}\ s'} \qquad (\textsc{GetNoLeak})$$

$$\frac{v \approx_k (l.\mathbf{get}\ s)}{l.\mathbf{put}\ v\ s \approx_k s} \qquad (\textsc{GetPut})$$

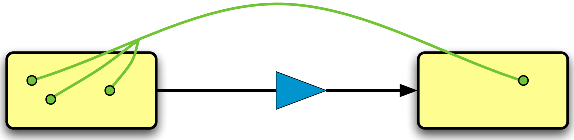
(See paper for a dynamic approach to integrity tracking.)

# Future Directions

# Data Provenance

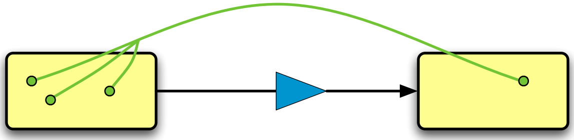


# Data Provenance



Provenance is metadata that describes the origin and causal history of pieces of data.

# Data Provenance

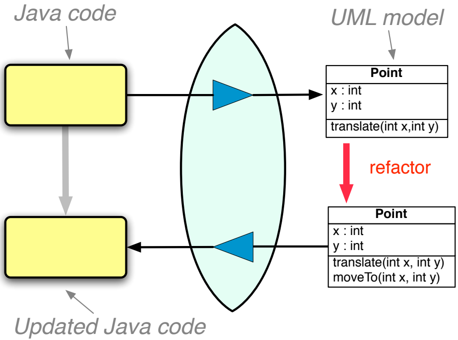


Provenance is metadata that describes the origin and causal history of pieces of data.

In the context of lenses, provenance is useful

- for fine-grained tracking of confidentiality and integrity [Foster, Green, Tannen PODS '08]
- for incremental view maintenance
- as an additional input to the **put** function

# Model Transformations




Much interest in the software engineering community in using lenses for bidirectional model transformations [Stevens '07] [Czarnecki, Foster, Hu, Lämmel, Schürr, Terwilliger ICMT '09]

Requires lenses for richer structures — e.g., graphs.

# Conclusion

"Bidirectional programming languages are an effective and elegant means of describing updatable views"

## Lenses

- Semantic space of well-behaved bidirectional transformations
- Provides foundation for bidirectional languages

## Boomerang

- Language for lenses on strings
- Natural syntax based on regular operators
- Extensions to handle ordered and ignorable data
- Type system guarantees well-behavedness and totality

## Implementation & Applications

- Lenses for a number of real-world formats
- Adoption in Augeas
- Updatable security views

# Thank You!

Collaborators: Benjamin Pierce, Alexandre Pilkiewicz, Aaron Bohannon, Michael Greenberg, and Alan Schmitt.

Want to play? Boomerang is available for download.

- Source code (LGPL)
- Precompiled binaries for Linux, OS X, Windows
- Research papers
- Tutorial and demos

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