Not so Homoiconic

EuroClojure 2012
Christophe Grand @cgrand

Source tooling is hard

Source tooling should not be a bunch of regexes

Source tooling?

- Basis of refactoring and assistants
  - Extracting fns
  - Maintaining ns forms and project.clj
  - Scaffolding defrecord/deftype/...
  - paredit
  - ...
- Source-as-text to source-as-text transformations

It’s the fault of homoiconicity!

What would a Java toolsmith do?*

*given enough money and time

org.eclipse.jdt.core.dom

- Bite the bullet
- 8 interfaces and 112 classes for code
- 1 interface and 6 classes for transformations

And it works!

But nobody wants to write new transfos

What would a Clojure toolsmith do?

```
(-> src read
transform pprint)
```

And it works!*

And anybody can write new transforms

And it works!*

And anybody can write new transforms

*As long as you are a compiler

WYSINWTRS

What You See Is Not What The Reader See

The reader ate my layout!

```
;; 2. collect new rules
(extend-protocol RuleFragment
  ;; a ref to another rule: add support for + ? or
* suffixes
  clojure.lang.Keyword
    (unsugar [kw]
      (if-let [[_ base suffix] (re-matches #"(.*?)
([+*?])" (name kw))]
        (unsugar [(keyword base) (keyword
suffix)])
        kw))
    (collect [this unspaced top-rulename]
      nil)
    (develop [this rewrite space]
      [[this]])

  ;; a vector denotes a sequence, supports postfix
operators :+ :? and :*
  clojure.lang.IPersistentVector
    (unsugar [this]
      (reduce #(condp = %2
                 :* (conj (pop %1) #{[] (Repeat+.
(peek %1))})
                 :+ (conj (pop %1) (Repeat+. (peek
%1)))
                 :? (conj (pop %1) #{[](peek %1)})
                 (conj %1 (unsugar %2))) [] this))
    (collect [items unspaced top-rulename]
      (mapcat #(collect % unspaced top-rulename)
items))
    (develop [items rewrite space]
      (reduce #(for [x (rewrite %2 space) sp space
xs %1]
                 (concat x (and (seq x) (seq xs)
sp) xs))
        [()] (rseq items))))
```

```
(extend-protocol
 RuleFragment
 clojure.lang.Keyword
 (unsugar
  [kw]
  (if-let
   [[_ base suffix] (re-matches #"(.*?)
([+*?])" (name kw))]
   (unsugar [(keyword base) (keyword suffix)])
   kw))
 (collect [this unspaced top-rulename] nil)
 (develop [this rewrite space] [[this]])
 clojure.lang.IPersistentVector
 (unsugar
  [this]
  (reduce
   (fn*
    [p1__5950# p2__5949#]
    (condp
     =
     p2__5949#
     :*
     (conj (pop p1__5950#) #{[] (Repeat+. (peek
p1__5950#))})
     :+
     (conj (pop p1__5950#) (Repeat+. (peek
p1__5950#)))
     :?
     (conj (pop p1__5950#) #{[] (peek p1__5950#)})
     (conj p1__5950# (unsugar p2__5949#))))
   []
   this))
 (collect
  [items unspaced top-rulename]
  (mapcat
   (fn* [p1__5951#] (collect p1__5951# unspaced
top-rulename))
   items))
 (develop
  [items rewrite space]
  (reduce
   (fn*
    [p1__5953# p2__5952#]
    (for
     [x (rewrite p2__5952# space) sp space xs
p1__5953#]
     (concat x (and (seq x) (seq xs) sp) xs)))
   [()]
```

Meikel (VimClojure)

Laurent (Counterclockwise)

The reader ate my layout

- Indentation/whitespaces
- Comments
- Maps and sets orderings
- Metadata (shortcuts and «stacks»)
- ::autoresolving/keyword

- ` ~ and ~@
- #(%1 %2)
- @ #’
- 0xFFFF, 42/8
- \u00E8 (\è)
- ...

But it’s *this* close!

It’s unreadable but it works!

Spoiled ~~kids~~ devs!

- Macros/sexprs transforms are easy
- Structural transforms should be as easy!
- Nobody wants to deal with JDT-like complexity
- We (I?) want macro coziness but character-perfect transformations!

To have a cake and eat it

Structural transforms

- Provide a function of sexprs to sexprs
  - Macro-like
- But transform the source (as text)
  - Preserve layout

What can be done?

Option 1: the reader

- A reader which keeps everything
  - Either it doesn't really keep everything
  - Still qualify as homoiconic? (parser)
  - Anyway, the complexity is passed down to the transformation writer

Option 2: the language

- Forbid some of the problematic stuff
- Shoehorn the remaining in language and or convention
    - Example : reclaim ^"string" from ^{:tag "string"} to ^{:doc "string"} to replace comments

Option 3:
Admit it, we have a view-update problem!

View-update problem

- A database thing
- How to update the table when the view is updated?
- Qualify a whole range of problems
  - GUI
  - File sync
    - Benjamin C. Pierce’s work on Unison and bidirectional programming

YAVUP

Yet Another View Update Problem

How to update the table when the view is updated?

YAVUP

Yet Another View Update Problem

source text

How to update the ~~table~~ when the view is updated?

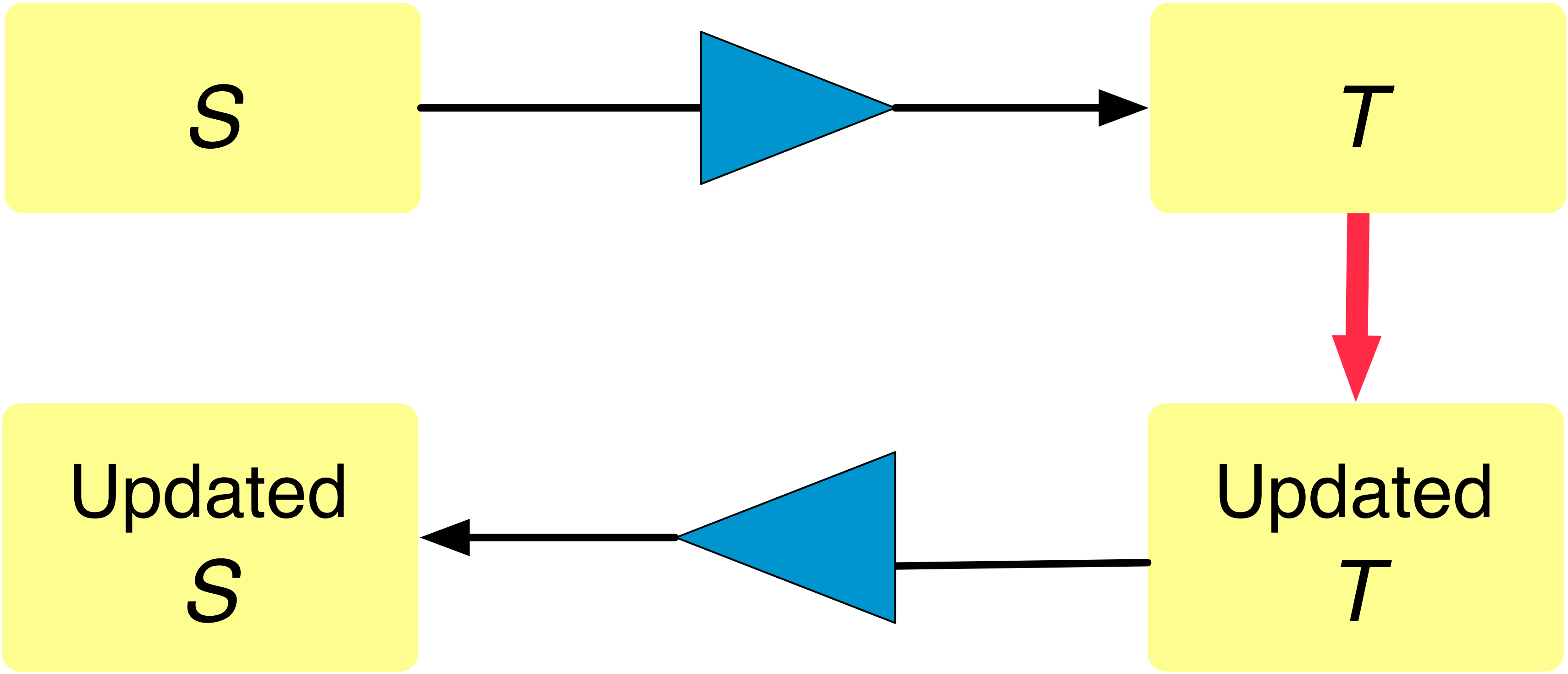
YAVUP

Yet Another View Update Problem

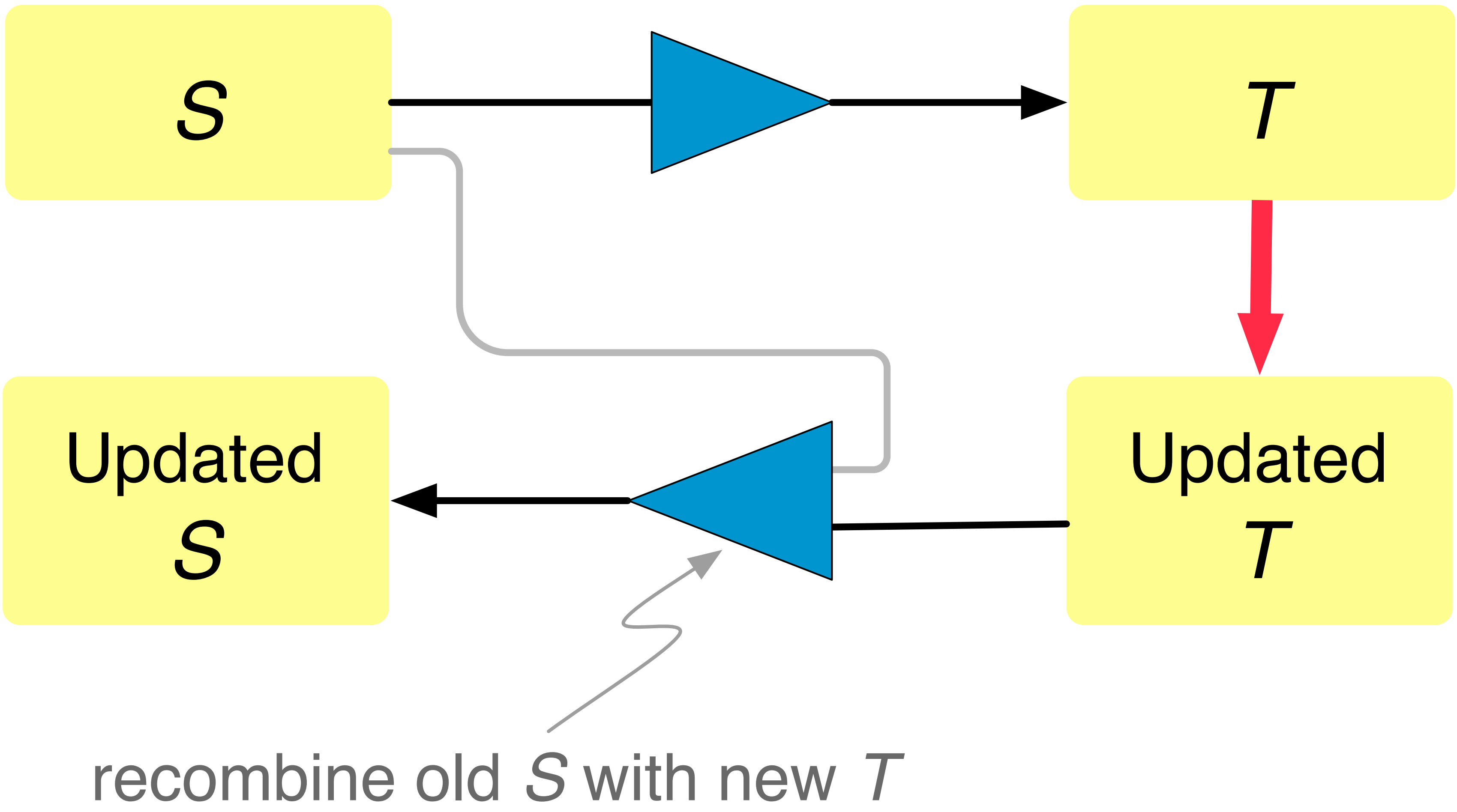
source text read value

How to update the ~~table~~ when the ~~view~~ is updated?

View update

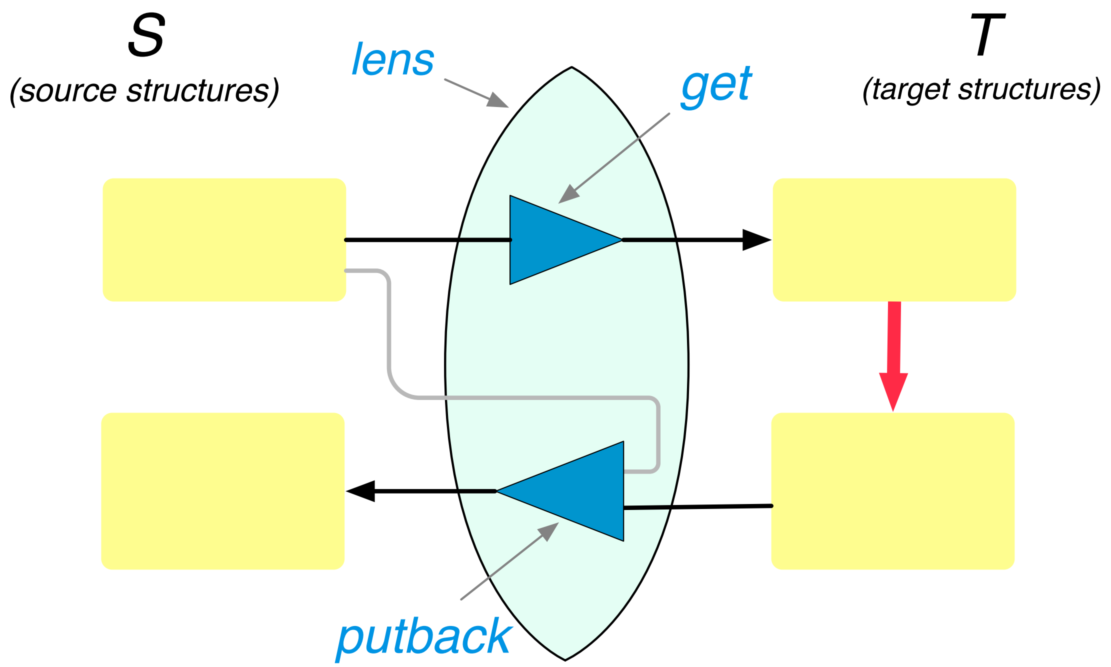


read is not bijective



recombine old *S* with new *T*

Bidir programming



- Introduce lenses
- Lens are composable
- Interesting idea
  - Plus Pierce studied the case of tree transformations!

Lenses all the way down

- Defining a view = combining lens into a bigger lens
- Define a view, get the putback for free!

Overthinking it

but the putback fn is a good idea

Let be pragmatic

- The «get» function isn’t going to change
    - It’s the reader!
    - No need for lenses and lens combinators
- «putback» is **the real good idea**
- Let’s handcraft the «putback» function
  (defn putback [expr src-as-txt] ...)

Master Plan

- Source -> parse tree
- Parse tree -> expr + translation log
- Expr -> Expr2
- Expr2 + translation log -> Parse tree 2
- Parse tree 2 -> Source 2

Master Plan

- Source -> parse tree Done!
- Parse tree -> expr + translation log
- Expr -> Expr2
- Expr2 + translation log -> Parse tree 2
- Parse tree 2 -> Source 2

Master Plan

- Source -> parse tree Done!
- Parse tree -> expr + translation log Easy!
- Expr -> Expr2
- Expr2 + translation log -> Parse tree 2
- Parse tree 2 -> Source 2

Master Plan

• Source -> parse tree	Done!
• Parse tree -> expr + translation log	Easy!
• Expr -> Expr2	User!
• Expr2 + translation log -> Parse tree 2	
• Parse tree 2 -> Source 2	

Master Plan

• Source -> parse tree	Done!
• Parse tree -> expr + translation log	Easy!
• Expr -> Expr2	User!
• Expr2 + translation log -> Parse tree 2	Putback
• Parse tree 2 -> Source 2	

Master Plan

• Source -> parse tree	Done!
• Parse tree -> expr + translation log	Easy!
• Expr -> Expr2	User!
• Expr2 + translation log -> Parse tree 2	Putback
• Parse tree 2 -> Source 2	Easy!

Parse tree vs sexpr

```
(z (a
    b)) (c
         d)
```

```
{:tag :net.cgrand.parsley/root,
 :content
 [{:tag :list,
   :content
   ["("
    {:tag :symbol, :content
[{:tag :name, :content ["z"]}]}
    {:tag :whitespace, :content [" "]}
    {:tag :list,
     :content
     ["("
      {:tag :symbol, :content
[{:tag :name, :content ["a"]}]}
      {:tag :newline, :content ["\n"]}
      {:tag :whitespace, :content ["
```

```
(z (a b)) (c d)
```

Translation log

- For each node of the parse-tree
  - remember the corresponding expression

Translation log

- For each ~~node~~ location of the parse-tree
  - remember the corresponding expression
- Locations allow to look around
  - find comments, indentation...

Record translation log

- Translation is straightforward
- Define the translation function as being ^:dynamic
- Rebind it to a logging self!
  - Beware of laziness

Translation log

- Scanned to find the closest parse-tree node corresponding to an expression
    - Prefer identity over value
    - Some similarity heuristics may be added
- When not found, render the expression using a default algorithm
    - pprint?

Indentation

- Locations give the whole context
  - Allow to compute indentation
  - Allow to adjust the indentation of code based on new context

Control freak

- Some transformations are only about layout
  - Just work directly on the parse tree
- Mixed transformations should be decoupled:
  - Structural transformations
  - Layout transformations

Demo

To be continued

- Similarity heuristics to recover more layout on updated nodes
- Support of range operations
- Support for splicing (unwrap)
- Integration in IDEs
    - Counterclockwise for a start
    - Standalone backend

Thank you!

Christophe Grand @cgrand