Fold and XML Transformation

Andy Wingo
0.1 origins
Pain avoidance, indignation Preparing for a presentation in February, I was struck with the realization: if I’m doing this in my free time, for fun, why force myself to endure the abomination known as OpenOffice?

0.2 svg instead of openoffice
Each layer can be a slide
(picture of inkscape with xml editor open)

0.3 bullets in svg is a drag
"This could be better"
SVG is XML, and I have a hammer!
(Here I knew that I had a problem to work on.)

0.4 simple slides language

    <slides>
    <slide>
        <title>Hi.</title>
        <para>Hello<br/>world</para>
    </slide>
    </slides>

0.5 example in sxml

    (slides
    (slide
        (title "Hi.")
        (para "Hello" (br) "world")))

0.6 try rewrite with pre-post-order
Table-driven rewrite of S-expressions
Great stuff
Kicks XSLT in the pants
Try pre-post-order to transform some simple slides vocab to SVG

0.7 pre-post-order: slides->html

    ‘((slides . ,(lambda (tag . kids)
            ‘(html (body ,@kids)))))
    (slide . ,(lambda (tag . kids)
            ‘(div (@ (class "slide"))
                ,@kids)))
    (title . ,(lambda (tag . kids)
            ‘(h1 ,@kids)))
0.8 slides as html

```html
(html
  (body
    (div (@ (class "slide"))
      (h1 "Hi."
       (p "Hello" (br) "world"))))
```

0.9 slides as svg

```svg
(svg (@ (width "1024") (height "768"))
  (g (text
      (@ (x "96") (y "216")
       (font-size "64px"))
      (tspan (@ (x "96") (y "216")) "Hello"
      (tspan (@ (x "96") (y "280")) "world")))
```

0.10 pre-post-order: slides->svg

```lisp
(tspan (@ (x "96") (y "216")) "Hello"
  (tspan (@ (x "96") (y "280")) "world")
```

(Here I knew I had an interesting problem.)

0.11 the problem

Rendering a declarative document into SVG is a context-sensitive transformation. Post-order transformation is context-insensitive.

0.12 multithreadedness

Post-order can be expressed in terms of the multithreaded foldt

```lisp
(define (foldt fup fhere tree)
  (if (atom? tree)
      (fhere tree)
      (fup (map (lambda (kid)
                  (foldt fup fhere kid))
             tree))))
```

0.13 layout is a single-threaded

Need new combinator in terms of foldts: monadic layout seed
(define (foldts fdown fup fhere seed tree)
  (if (atom? tree)
      (fhere seed tree)
      (fup seed
       (fold (lambda (kid kseed)
              (foldts fdown fup fhere
                  kseed kid))
              (fdown seed tree)
              tree)
       tree)))

0.14 macro expansion for xml
pre-post-order can also do pre-order rewrites of the tree
Need ability to modify tree being traversed

0.15 solution: foldts*

(define (foldts* fdown fup fhere seed tree)
  ...
  (call-with-values
   (lambda () (fdown seed tree))
   (lambda (kseed tree)
     (fup seed
      ...))))

0.16 multi-valued seeds painful
Writing foldts* handlers painful
Need automatic destructuring of seed
Solution: multi-valued fold
• Idea taken from scsh

0.17 foldts*-values
Analogous to fold-values:

(define (fold-values proc list . seeds)
  (if (null? list)
      (apply values seeds)
      (call-with-values
       (lambda ()
         (apply proc (car list) seeds))
       (lambda seeds
         (apply fold-values
           proc (cdr list) seeds))))))
0.18 **foldts*-values**
A general traversal combinator
   Handlers convenient to write, easy destructuring of multi-valued seed
   Efficient

0.19 **pre-post-order for svg layout?**
The svg problem: deriving domain-specific combinators on top of foldts*-values
   foldts not terribly nice to program directly
   "fold-layout"

0.20 **building on foldts*-values**
   - Decide the format for the seeds
   - Implement fdown, fup, fhere

0.21 **fold-layout seed format**
   - return value
   - some representation of "layout"
   - hierarchical params
   - current bindings table
   - "post-handler"

0.22 **fold-layout bindings example**

\'((slide
   (pre-layout . ,slide-pre-layout)
   (post . ,slide-post))
   (header
   (post . ,header-post))
   (cartouche
   (pre-layout . ,cartouche-pre-layout)
   (post . ,cartouche-post))
   (p
   (post . ,p-post))
   (*text* . ,text-handler))

0.23 **fold-layout: implementing fdown**
Handlers to call in fdown: pre-layout, pre/macro
   (define (cartouche-pre-layout
   tree params layout)
   (let-layout layout (x y)
   (let-params params (margin-left
   margin-top))
0.24 fold-layout: implementing fup

Handlers to call in fup: post

(define (p-post tag params old-layout layout kids)
  (values
   layout
   '(text
     (@ (x ,(make-text-x params old-layout))
       (y ,(make-text-y params old-layout)))
     ,@kids)))

0.25 fold-layout: implementing fhere

Handlers to call in fhere: *text*

(define (text-handler text params layout)
  (values
   (layout-advance-text-line params layout)
   '({text
     (@ (x ,(make-text-x params layout))
       (y ,(make-text-y params layout)))
     ,text))))

0.26 conclusions (1/2)

- foldts underlies (all?) XML transformations
- foldts* is like foldts, but allows macro transformation
  - foldts*-values is a convenient foldts*

0.27 conclusions (2/2)

- When you need foldts, you generally want a domain-specific combinator built on foldts.
  - It is possible to "derive" such combinators methodically
- fold-layout is such a combinator
  - Graphics layout with functional programming

0.28 questions?

Thanks for listening!

Andy Wingo
wingo@pobox.com
wingolog.org/software/guile-present/