origins

Pain avoidance, indignation
svg instead of openoffice

Each layer can be a slide

Why Scheme?

Not because it's common

Not because your boss tells you to

Not because your friends are doing it
bullets in svg is a drag

"This could be better"

SVG is XML, and I have a hammer!
simple slides language

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

(slides
  (slide
    (title "Hi.")
    (para "Hello" (br) "world")))
try rewrite with pre-post-order

Table-driven rewrite of S-expressions

Great stuff
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)))
(*text*  . ,(lambda (tag text)
      text))
...

slides as html

(html
  (body
    (div (@ (class "slide"))
       (h1 "Hi.")
       (p "Hello" (br) "world"))))
slides as 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")))
pre-post-order: slides->svg

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

?
the problem

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

Post-order transformation is context-insensitive.
multithreadedness

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

(define (foldt fup fhere tree)
  ...      (fhere tree)
      (fup (map (lambda (kid)
                  (foldt fup fhere kid))
                tree))))
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)))
macro expansion for xml

pre-post-order can also do pre-order
rewrites of the tree

Need ability to modify tree being traversed
solution: foldts*

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

Writing foldts* handlers painful

Need automatic destructuring of seed

Solution: multi-valued fold

* Idea taken from scsh
foldts*-values

Analogous to fold-values:

(define (fold-values proc list . seeds)
  (if (null? list)
      (apply values seeds)
        (lambda seeds
          (apply fold-values
                 proc (cdr list) seeds)))))
foldts*-values

A general traversal combinator

Handlers convenient to write, easy destructuring of multi-valued seed

Efficient
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"
building on foldts*-values

* Decide the format for the seeds
* Implement fdown, fup, fhere
fold-layout seed format

* return value
* some representation of "layout"
* hierarchical params
* current bindings table
* "post-handler"
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))
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)
      (make-layout (+ x margin-left)
                   (+ y margin-top))))
fold-layout: implementing fup

Handlers to call in fup: post

(define (p-post tag params old-layout layout
              ...   `(text
     (@ (x ,(make-text-x params old-layout))
        (y ,(make-text-y params old-layout)))
     ,@kids)))
fold-layout: implementing fhere

Handlers to call in fhere: *text*

(define (text-handler text params layout)
  (values
    (layout-advance-text-line params layout)
    '(tspan
      (@ (x ,(make-text-x params layout))
         (y ,(make-text-y params layout)))
      ,text)))
conclusions (1/2)

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

Thanks for listening!

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