PROGRAMACIÓN FUNCIONAL

Técnicas de diseño

Técnicas de diseño

- Primera Parte:
  - Técnica de Combinadores
  - Ejemplo: pretty printing
- Segunda Parte:
  - HyCom (Hypermedia Combinators)
    - Descripción de hypermedias en Haskell

Técnica de combinadores

- ¿Cómo lograr un diseño modular?
  - Empezar con módulos pequeños
  - Componerlos adecuadamente
- ¿Podríamos lograr expresar la composición de 'módulos' como si fuera un esquema?
  - Requiere expresar:
    - los 'módulos' como datos
    - la composición de `módulos' como una función

Técnica de combinadores

- Esta técnica
  - se aplica a dominios específicos
    - parsing (análisis sintáctico)
    - pretty printing (impresión con formato)
    - GUIs (construcción de interfases gráficas)
    - hypermedia (construcción de hyperdocumentos)
  - consiste en
    - definir un tipo para representar soluciones al problema
    - definir funciones para representar la combinación de soluciones, obteniendo nuevas soluciones
    - definir funciones que transformen las soluciones

Técnica de combinadores

- Ejemplo: pretty printing
  - Objetivo: mostrar datos estructurados
  - Considerar data Tree = Node String Tree Tree | Leaf

    ¿Cómo mostrar Node "primero" (Node "segundo" Leaf Leaf) (Node "tercero" Leaf Leaf)?
  - Lo ideal sería

    Node "primero" (Node "segundo" Leaf Leaf)
    (Node "tercero" Leaf Leaf)
  - pero ¡un nodo se muestra de una forma u otra, dependiendo del contexto!

Técnica de combinadores

- Aplicando la técnica al *pretty printing*

```
data Doc = ... -- el tipo de los documentos pretty
text :: String -> Doc
(<>) :: Doc -> Doc -> Doc  -- composición horizontal
($$) :: Doc -> Doc -> Doc  -- composición vertical
```

- ¿Cómo se mostraría el ejemplo previo?

```
text "Node \"primero\" " <>
    (text "Node \"segundo\" Leaf Leaf" $$
     text "Node \"tercero\" Leaf Leaf")
```

- Sin embargo, esto sólo permite estructuras fijas...

Técnica de combinadores

- Agregamos un combinador más
  - composición horizontal o vertical, según el contexto

    sep :: [Doc] -> Doc
- ¿Cómo se mostraría cualquier árbol Tree?

```
pp :: Tree -> Doc
pp Leaf = text "Leaf"
pp (Node s t1 t2) = text ("Node "++s) <>
                                sep [ parenpp t1, parenpp t2 ]
parenpp Leaf = pp Leaf
parenpp t = text "(" <> pp t <> text ")"
```

Técnica de combinadores

- Terminamos con otros combinadores primitivos
  - indentación dependiente del contexto
    nest :: Int -> Doc -> Doc
  - transformación de un Doc a String
    pretty :: Int -> Int -> Doc -> String
- ¿Cómo funciona nest?
  sep [text "while (x>0) do", nest 2 (text "x := x-2")]

  se vería en vertical vs. horizontal
  "while (x>0) do vs. "while (x>0) do x := x-2"
  x := x-2"

Técnica de combinadores

- Resumiendo: combinadores de *pretty printing*

```
data Doc = ... -- el tipo de los documentos pretty
text :: String -> Doc
(<>) :: Doc -> Doc -> Doc        -- horizontal
($$) :: Doc -> Doc -> Doc        -- vertical
sep :: [ Doc ] -> Doc            -- horizontal o vertical
nest :: Int -> Doc -> Doc        -- indentación sensible
pretty :: Int -> Int -> Doc -> String
```

Técnica de combinadores

- Otros combinadores de *pretty printing* (definidos)

```
(<+>) :: Doc -> Doc -> Doc
a <+> b = a <> text " " <> b

($?$) :: Doc -> Doc -> Doc        -- vertical
a $?$ b = sep [ a, b ]

empty :: Doc
empty = text  ""

vert, hor :: [ Doc ] -> Doc
vert = foldr ($$) empty
hor = foldr (<>) empty
```

Técnica de combinadores

- ¿Cómo implementar una biblioteca de combinadores?
  - por inspiración (la forma 'clásica')
  - reutilizando/adaptando otras bibliotecas de combinadores
  - estableciendo las propiedades de los combinadores y derivando el código (método Hughes)
  - especificando formalmente los combinadores y utilizando un generador automático de código (método Swierstra)

HyCom

- HyCom = Hypermedia Combinators
  - biblioteca de funciones para construir hyperdocumentos en Haskell
  - realizada por dos alumnos como tesis de Lic.
  - permite
    - utilizar métodos formales de diseño, sin perder expresividad
    - expresar los diseños de manera independiente de la plataforma de implementación
    - generar prototipos automáticamente (por ahora en HTML; más adelante, LaTeX, WML, SGML, ASP, etc.)

HyCom

- Idea básica:
  - definir la noción de *componente*
  - definir *combinadores* sobre componentes
  - definir *transformadores* sobre componentes
- Tres niveles de componentes:
  - componentes de *datos* (e.g. personas)
  - componentes *navegacionales* (e.g. nodos, links)
  - componentes de *interface* (e.g. imágenes, botones)

HyCom

- Existen 2 versiones de HyCom
  - la versión 1
    - se concentra en combinadores de interface
    - no provee comunicación con otras bibliotecas
  - la versión 2 (descrita en la tesis) provee
    - generalidad y modificabilidad en todas las fases (mediante una jerarquía de *clases* de tipos)
    - mecanismos predefinidos de mapeo entre fases
    - persistencia de datos (mediante un mecanismo de comunicación con bases de datos ODBC)
    - programación CGI (mediante una biblioteca de funciones previamente definida)

HyCom

- Ejemplo (usando la versión 1 por simplicidad)
  - modelado de datos
    - datos representados como registros
    - constantes definidas en el script
  - modelado de navegación
    - constantes definidas en el script
    - funciones que agregan links
  - interface
    - funciones que obtienen una interface a partir de los datos
  - prototipación automática
    - funciones que generan los archivos HTML necesarios

HyCom

- Modelado de datos

```
data Imagen = IMG [String] String String

fidel = IMG ["Fidel, ", "en la escuela AFP."]
            "photos/fidelsmall.jpg"
            "photos/fidel.jpg"

someText =

  "Este es un ejemplo de uso de HyCom. Puede verse el
   fuente aquí. El texto y la imagen se generaron con la
   función 'homeHG'."
```

HyCom

- Modelado de navegación

```
httpLink :: NodeId -> String -> HL

fileLink :: NodeId -> String -> HL

nodeId :: String -> NodeId

myNd = nodeId "Test"

linksForText =
   [ ("HyCom.", httpLink myNd
                    "www-lifia.info.unlp.edu.ar/hycom/")
   , ("aquí.", fileLink myNd (addPath "prueba.hs"))
   ]
```

HyCom

- Modelado de interface - funciones predefinidas

```
imgDumbHG :: String -> HG
orgAnchorHG :: HG -> HL -> HG
(/=\) :: HG -> HG -> HG  -- Composición vertical
(<=<) :: HG -> HG -> HG  -- Composición horizontal
txtHG :: String -> HG
alignCenter :: HG -> HG
asTitle :: Int -> HG -> HG
txtWithAnchors :: String -> [(String, HL)] -> HG
hSpaceHG :: Int -> HG
divisionLine :: HG
```

HyCom

- Modelado de interface - elementos de interface

```
someTextHG = txtWithAnchors someText linksForText
imageHG nodeId (IMG ts imgsmall img) =
        orgAnchorHG (imgDumbHG (imgsmall))
                            (fileLink nodeId (addPath img))
          /=\
        foldr1 (/=\) (map txtHG ts)
header title = alignCenterHG (asTitle 1 (txtHG title))
vBarHG = hSpaceHG 50                                    <=<
                imgDumbHG ("imgs/vcolorbar.gif") <=<
                hSpaceHG 50
```

HyCom

- Modelado de interface - interface del nodo

```
backgroundImg :: String -> Node -> Node
nodes = [ node1 ]
node1 = backgroundImg "photos/back-fidel.jpg" $
            node myNd homeHG
homeHG = header "Página de Test"     /=\
                divisionLine                    /=\
                (someTextHG
                        <=<  vBarHG  <=<
                 imageHG myNd fidel
                )
```

HyCom

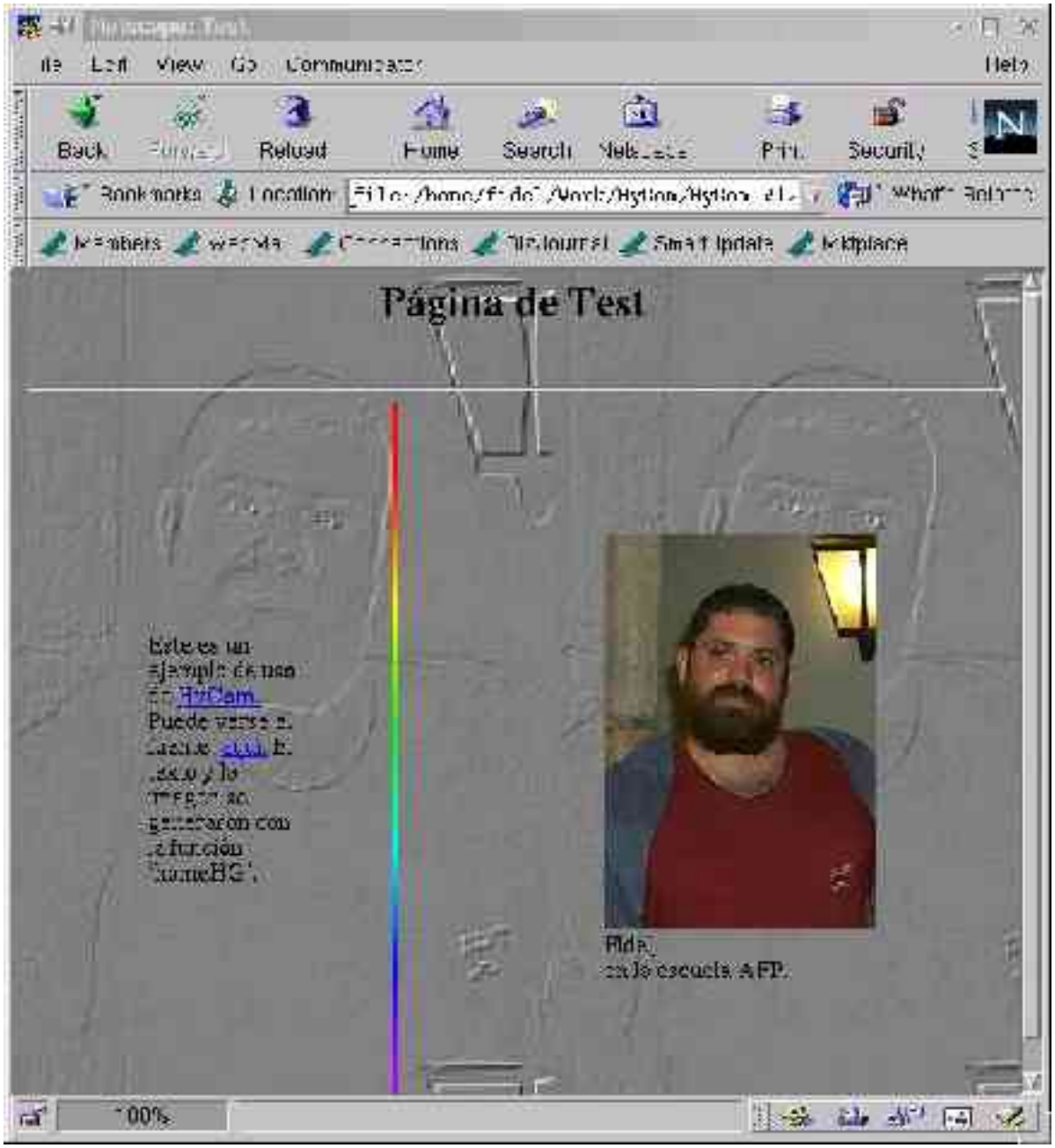
- Prototipación

```
hyComToHTML :: [ (HD, PageNames) ]
hd :: [ Node ] -> HD
type PageNames = [ (NodeId, String) ]

main =  hyComToHTML [ (doc, htmlFiles) ]

htmlFile :: PageNames
htmlFiles = [ (myNd, "prueba.html") ]

doc :: HD
doc = hd nodes
```


Página de Test

Resumen

- La técnica de combinadores
  - permite la fácil solución de problemas en un dominio específico
  - guía la construcción de una biblioteca de funciones para tales problemas
- HyCom
  - utiliza la técnica de combinadores
  - permite expresar documentos hypermediales
  - permite prototipación rápida y sencilla