DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

# Trabajo Práctico Nº1

**Programación Funcional.**

8 de septiembre de 2015 Paradigmas de Lenguajes de Programación.



| Integrante | LU | Correo electrónico |
|---|---|---|
| Ferranti, Marcelo | 744/08 | mferranti89@gmail.com |
| Pizzagalli, Matías | 257/12 | matipizza@gmail.com |
| Rodriguez, Pedro | 197/12 | pedro3110.jim@gmail.com |



| Instancia | Docente | Nota |
|---|---|---|
| Primera entrega | | |
| Segunda entrega | | |





**Facultad de Ciencias Exactas y Naturales**
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (++54 +11) 4576-3300
http://www.exactas.uba.ar

# 1. Routes.hs

```haskell
-- Routes.hs --
module Routes where

import Data.List
import Data.Maybe

data PathPattern = Literal String | Capture String deriving (Eq, Show)

data Routes f = Route [PathPattern] f | Scope [PathPattern] (Routes f) | Many [Routes f] deriving Show

-- Ejercicio 1: Dado un elemento separador y una lista, se debera partir la lista en sublistas de
    acuerdo a la aparici\'on del separador (sin incluirlo).
--              Para contemplar el caso base que se nos pide, debemos devolver [] si nos pasan la lista
    vacia.
--              Si la lista que nos pasan no es vacia, la recorremos y cada vez que aparece un elemento
    igual al que nos pasan, agrego la sublista que
--              estaba generando en ese momento para la solucion final, y el resto de las sublistas las
    voy a agregar recursivamente a partir de la
--              porcion de lista aun no recorrida.
split :: Eq a => a -> [a] -> [[a]]
split n xs = if length xs /= 0 then split' n xs else []

split':: Eq a => a -> [a] -> [[a]]
split' n = foldr (\x r -> if (x/=n) then (\yss -> (x:head yss):(tail yss)) r else [[]] ++ r) [[]]

-- Ejercicio 2: A partir de una cadena que denota un patr\'on de URL se deber\'a construir la secuencia
    de literales y capturas correspondiente.
--              Como 'split' devuelve cadenas vacias cuando hay uso inadecuado de las barras "//",
    procesamos la particion del string a partir de
--              la barra para tener una lista de literales y/o capturas representados de forma
    aceptable. A partir de esta lista de strings no
--              vacios podemos generar el [PathPattern] que se pide.
pattern :: String -> [PathPattern]
pattern xs = pattern' $ (foldr (\x r -> if x=="" then r else x:r) []) (split '/' xs)

pattern' :: [String] -> [PathPattern]
pattern' = (foldr (\x r -> if ((not (null x)) && (head x) == ':') then (Capture (tail x)):r
                  else (Literal x):r) [])

-- Ejercicio 3: Obtiene el valor registrado en una captura determinada. Se puede suponer que la captura
    est\'a definida en el contexto.
--              A raiz de esta suposicion, sabemos que habra algun elemento del PathContext que se
    corresponda con el string que nos pasan.
--              Luego, cuando encontramos ese string devolvemos a su pareja.
type PathContext = [(String, String)]

get :: String -> PathContext -> String
get s = foldr (\(key,value) r -> if key==s then value else r) s

-- Ejercicio 4: Dadas una ruta particionada y un patr\'on de URL, trata de aplicar el patr\'on a la
    ruta y devuelve, en caso de que
--              la ruta sea un prefijo v\'alido para el patr\'on, el resto de la ruta que no se haya
    llegado a consumir y el contexto capturado
--              hasta el punto alcanzado. Se puede usar recursi\'on expl\'icita.

-- unir : lo usamos para agregar adelante del PathContext resultante del procesamiento de una ruta
--        el resultado de haber procesado algun elemento de la misma. Lo utilizamos como funcion
    auxiliar de 'matches'
unir :: PathContext -> Maybe ([String], PathContext) -> Maybe ([String], PathContext)
unir pc Nothing = Nothing
unir pc (Just (ss,xs)) = Just (ss,pc++xs)

-- matches : asumimos que si el PathContext tiene pide mas informacion de la que la ruta
--           nos puede brindar, devolvemos Nothing. A los literales y capturas los consumimos, y
--           a las capturas ademas las procesamos y agregamos adelante en el resultado final, pues son
    los datos que debemos guardar.
--           (Se permite usar recursion explicita)
matches :: [String] -> [PathPattern] -> Maybe ([String], PathContext)
matches ss [] = Just (ss, [])
matches [] xs = Nothing
matches (s:ss) ((Literal x):xs) = if s==x then (matches ss xs) else Nothing
matches (s:ss) ((Capture x):xs) = unir [(x,s)] (matches ss xs)

-- DSL para rutas
route :: String -> a -> Routes a
route s f = Route (pattern s) f

scope :: String -> Routes a -> Routes a
scope s r = Scope (pattern s) r

many :: [Routes a] -> Routes a
many l = Many l

-- Ejercicio 5: Definir el fold para el tipo Routes f y su tipo. Se puede usar recursi\'on expl\'icita.
-- Routes f = Route [PathPattern] f | Scope [PathPattern] (Routes f) | Many [Routes f]
foldRoutes :: ([PathPattern] -> b -> c) -> ([PathPattern] -> c -> c) -> ([c] -> c) -> Routes b -> c
foldRoutes fRoute fScope fMany route = case route of
    Route xs f  -> fRoute xs f
    Scope xs r  -> fScope xs (rec $ r)
    Many  lr    -> fMany (map rec lr)
    where rec = foldRoutes fRoute fScope fMany


-- Auxiliar para mostrar patrones. Es la inversa de pattern.
patternShow :: [PathPattern] -> String
patternShow ps = concat $ intersperse "/" ((map (\p -> case p of
  Literal s -> s
  Capture s -> (':':s)
  )) ps)


-- Ejercicio 6: Genera todos los posibles paths para una ruta definida.
--              caso Route: devolvemos la unica ruta que se puede generar
```

```
--                caso Scope: a cada ruta generada en la recursion  sobre la ruta que toma Scope como
     parametro, se le agrega adelante
--                              el string patron que Scope considera
--                caso Many: como son todos resultados independientes, la concatenacion de estos son
     todos los paths que se pueden generar
paths :: Routes a -> [String]
paths = foldRoutes (\xs f -> [patternShow xs])
                   (\xs r -> map (\x -> if x==[] then ((patternShow xs) ++ x) else (((patternShow xs)
                        ++ "/") ++ x)) r)
                   (\xs -> concat xs)

-- Ejercicio 7: Eval\'ua un path con una definici\'on de ruta y, en caso de haber coincidencia, obtiene
     el handler correspondiente
--                y el contexto de capturas obtenido.
{-
Nota: la siguiente funci\'on viene definida en el m\'odulo Data.Maybe.
 (=<<) :: (a->Maybe b)->Maybe a->Maybe b
 f =<< m = case m of Nothing -> Nothing; Just x -> f x
-}

-- eval : Primero, trabajamos con el path en su forma partida, para mayor prolijidad del codigo
--        caso Route [PathPattern] f : solo en caso de que el path considerado (ss) alcance para
     consumir todo el [PathPattern] considerado (xs),
--              (esto lo chequeamos con el matches ss xs) y si el path en consumido en su totalidad (
     null x), entonces devolvemos la funcion
--              correspondiente a la evaluacion de dicho path.
--        caso Scope [PathPattern] (Routes f) : primero se chequea que el path ss que nos pasan matchee
      con el [PathPattern] xs. Esto lo hacemos
--              con "matches ss xs". Si esto sucede, nos fijamos si la porcion del path ss que no se
     pudo consumir con xs (en el codigo, la x),
--              si esa porcion de string matchea con la ruta considerada dentro del Scope. Esto lo
     chequeamos con r (notar que
--              fr :: [String] -> Maybe (a, PathContext)). Si efectivamente, en la recursion se logra
     matchear lo que queda del string ss con
--              cierta ruta, devolvemos la funcion correspondiente al desarrollo de dicha ruta, y
     concatenamos al PathContext generado por
--              esa ruta al PathContext que habiamos obtenido al consumir el path ss que nos habian
     pasado. Esto lo hacemos con y++y', donde y es lo
--              consumido por ss antes de entrar a Scope y y' lo consumido por ss ya dentro de la ruta
     del Scope.
--        caso Many [Routes f] : se tiene una lista de resultados, candidatos a matchear con el path
     que nos pasan. si alguno de estos
--              matchea con el path ss, tomamos ese como resultado. Si no matchea con ninguno, devuelve
     Nothing pues quiere decir que el path
--              ss no se corresonde con ninguno de los resultados disponibles. Notar que fx :: String ->
      Maybe (a,PathContext)

eval :: Routes a -> String -> Maybe (a, PathContext)
eval route s = eval' route (split '/' s)

eval' :: Routes a -> [String] -> Maybe (a, PathContext)
eval' = foldRoutes (\xs f  -> (\ss -> (\(x,y) -> if null x then Just (f,y) else Nothing) =<< (matches
     ss xs)))
                   (\xs fr -> (\ss -> (\(x,y) -> (\(x',y') -> Just (x', y++y')) =<< (fr x)) =<< (
                        matches ss xs)))
                   (\lr    -> (\ss -> (foldr (\fx r -> if isNothing (fx ss) then r else (fx ss))
                        Nothing) lr))

-- Ejercicio 8: Similar a eval, pero aqu\'i se espera que el handler sea una funci\'on que recibe como
     entrada el contexto
--              con las capturas, por lo que se devolver\'a el resultado de su aplicaci\'on, en caso de
      haber coincidencia.
--              Es evidente: si se obtiene una funcion al evaluar path, se aplica dicha funcion sobre
     el PathContext correspondiente
--              a la evaluacion de path.
exec :: Routes (PathContext -> a) -> String -> Maybe a
exec routes path = (\px -> Just ((fst px) (snd px))) =<< (eval routes path)

-- Ejercicio 9: Permite aplicar una funcion sobre el handler de una ruta. Esto, por ejemplo, podr\'ia
     permitir la ejecuci\'on
--              concatenada de dos o m\'as handlers.
--              No hace falta explicarlo
wrap :: (a -> b) -> Routes a -> Routes b
wrap f = foldRoutes (\xs g -> Route xs (f g))
                    (\xs r -> Scope xs r)
                    (\lr   -> Many lr)

-- Ejercicio 10: Genera un Routes que captura todas las rutas, de cualquier longitud. A todos los
     patrones devuelven el mismo valor.
--              Las capturas usadas en los patrones se deber\'an llamar p0, p1, etc.
--              En este punto se permite recursi\'on expl\'icita.
catch_all :: a -> Routes a
catch_all h = undefined
```

# 2. SampleTests.hs

```
-- SampleTests.hs --
import Routes
import Test.HUnit
import Data.List (sort)
import Data.Maybe (fromJust, isNothing     )

rutasFacultad = many [
  route ""                "ver_inicio",
  route "ayuda"           "ver_ayuda",
  scope "materia/:nombre/alu/:lu" $ many [
    route "inscribir"     "inscribe_alumno",
    route "aprobar"       "aprueba_alumno"
  ],
  route "alu/:lu/aprobadas"  "ver_materias_aprobadas_por_alumno"
  ]

rutasStringOps = route "concat/:a/:b" (\ctx -> (get "a" ctx) ++ (get "b" ctx))

-- evaluar t para correr todos los tests
--t = runTestTT allTests

allTests = test [
    "patterns" ~: testsPattern,
    "get" ~: testsGet,
    "matches" ~: testsMatches,
    "paths" ~: testsPaths,
    "eval" ~: testsEval,
    "evalWrap" ~: testsEvalWrap,
    "evalCtxt"~: testsEvalCtxt,
    "execEntity" ~: testsExecEntity,
    "exec" ~: testsExec
    ]

splitSlash = split '/'

testsPattern = test [
  splitSlash "" ~=? [],
    splitSlash "/" ~=? ["",""],
    splitSlash "/foo" ~=? ["", "foo"],
    splitSlash "foo/bar/plp/teorica/lambda" ~=? ["foo","bar","plp","teorica","lambda"],
    splitSlash "bar/foo//tp1/tests/" ~=? ["bar","foo","","tp1","tests",""],
    splitSlash "/bar/foo//tp1/tests" ~=? ["","bar","foo","","tp1","tests"],
    splitSlash "default/:index/user/?=new/create/valid?" ~=? ["default",":index","user","?=new","create
        ","valid?"],
    pattern "" ~=? [],
    pattern "/" ~=? [],
    pattern "lit1/:cap1/:cap2/lit2/:cap3" ~=? [Literal "lit1", Capture "cap1", Capture "cap2", Literal
        "lit2", Capture "cap3"],
    pattern "/alu/:lu/aprobadas" ~=? [Literal "alu", Capture "lu", Literal "aprobadas"],
    pattern "/alu/:lu//aprobadas//" ~=? [Literal "alu", Capture "lu", Literal "aprobadas"],
    pattern "foo/:bar/:plp/:dc/user/lambda" ~=?  [Literal "foo", Capture "bar", Capture "plp", Capture
        "dc", Literal "user", Literal "lambda"],
    pattern "lit1" ~=? [Literal "lit1"],
    pattern ":cap2" ~=? [Capture "cap2"]
    ]

testsGet = test [
    get "nombre" [("nombre","plp"),("lu","007-1")] ~=? "plp",
    get "a" [("a","b"),("a","c")] ~=? "b"
    ]

testsMatches = test [
    Just (["tpf"],[("nombreMateria","plp")]) ~=? matches (splitSlash "materias/plp/tpf") (pattern "
        materias/:nombreMateria"),
    matches (splitSlash "materia/plp/alu/007-01") (pattern "materia/:nombre") ~=? Just (["alu","007-01"
        ],[("nombre","plp")]),
    matches (splitSlash "user/pepe/profile/v1") (pattern "user/:name/profile/:api") ~=? Just ([],[("
        name","pepe"),("api","v1")]),
    matches [] [Literal "algo"] ~=? Nothing,
    matches (splitSlash "alu/007-01") (pattern "alumno/materia/:lu") ~=? Nothing
    ]

path0 = route "foo" 1
path1 = scope "foo" (route "bar" 2)
path2 = scope "foo" (route ":bar" 3)
path3 = scope "" $ scope "" $ many [ scope "" $ route "foo" 1]

testsEvalCtxt = test [
    Just (1, []) ~=? eval path0 "foo",
    Just (2, []) ~=? eval path1 "foo/bar",
    isNothing (eval path1 "foo/bla") ~? "",
    Just (3, [("bar", "bla")]) ~=? eval path2 "foo/bla",
    Just (1, []) ~=? eval path3 "foo",
    eval (route "foo/:lu" "ver_foo_de_alumno") "foo/001" ~=? Just("ver_foo_de_alumno", [("lu","001")]),
    eval (scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?")) "materia/plp/aprobada?"
        ~=? Just("la_materia_se_aprobo?", [("nombre","plp")]),
    eval (many [
            route "foo/:lu" "ver_foo_de_alumno",
            scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?"),
            scope "alumno/:lu" (many [route "registrar" "registrar_alumno", route "editar" "editar_
                alumno", route "eliminar" "eliminar_alumno"])
            ]) "alumno/001/editar" ~=? Just("editar_alumno",[("lu","001")]),
    eval (route "foo/:lu" "ver_foo_de_alumno") "foo/alu/001" ~=? Nothing,
    eval (scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?")) "materia/nombre/plp/
        aprobada?" ~=? Nothing,
    eval (many [
            route "foo/:lu" "ver_foo_de_alumno",
            scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?"),
            scope "alumno/:lu" (many [route "registrar" "registrar_alumno", route "editar" "editar_
                alumno", route "eliminar" "eliminar_alumno"])
            ]) "foo/001/editar" ~=? Nothing
    ]
```

```
path4 = many [
  (route "" 1),
  (route "lo/rem" 2),
  (route "ipsum" 3),
  (scope "folder" (many [
    (route "lorem" 4),
    (route "ipsum" 5)
    ]))
  ]


testsEval = test [
        1 ~=? justEvalP4 "",
        4 ~=? justEvalP4 "folder/lorem"
    ]
    where justEvalP4 s = fst (fromJust (eval path4 s))

path410 = wrap (+10) path4

testsEvalWrap = test [
        14 ~=? justEvalP410 "folder/lorem",
        eval (wrap reverse (route "foo/:lu" "ver_foo_de_alumno")) "foo/001" ~=? Just("onmula_ed_oof_rev
             ",[("lu","001")]),
        eval (wrap (\f -> length f) (scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?"
             ))) "materia/plp/aprobada?" ~=? Just(length "la_materia_se_aprobo?",[("nombre","plp")]),
        eval (wrap reverse rutasFacultad) "ayuda" ~=?  Just("aduya_rev",[]),
        eval (wrap reverse (route "foo/:lu" "ver_foo_de_alumno")) "foo/lu/001" ~=? Nothing,
        eval (wrap (\f -> length f) (scope "materia/:nombre" (route "aprobada?" "la_materia_se_aprobo?"
             ))) "materia/plp/aprobada" ~=? Nothing,
        eval (wrap reverse rutasFacultad) "ayuda/inscribir" ~=?  Nothing
    ]
    where justEvalP410 s = fst (fromJust (eval path410 s))


-- ejemplo donde el valor de cada ruta es una funci\'on que toma context como entrada.
-- para estos se puede usar en lugar adem\'as de eval, la funci\'on exec para devolver
-- la applicaci\'on de la funci\'on con sobre el contexto determinado por la ruta
rest entity = many [
  (route entity (const (entity++"#index"))),
  (scope (entity++"/:id") (many [
    (route "" (const (entity++"#show"))),
    (route "create" (\ctx -> entity ++ "#create_of_" ++ (get "id" ctx))),
    (route "update" (\ctx -> entity ++ "#update_of_" ++ (get "id" ctx))),
    (route "delete" (\ctx -> entity ++ "#delete_of_" ++ (get "id" ctx)))
    ]))
  ]

path5 = many [
  (route "" (const "root_url")),
  (rest "post"),
  (rest "category")
  ]

testsPaths = test [
    sort ["","post","post/:id","post/:id/create","post/:id/update","post/:id/delete","category","
         category/:id","category/:id/create","category/:id/update","category/:id/delete"] ~=? sort (
         paths path5),
    paths (route "hola" "ver_hola") ~=? ["hola"],
    paths (route "" "ver_home") ~=? [""],
    paths (scope "foo/:bar" (route "hola" "ver_hola")) ~=? ["foo/:bar/hola"],
    paths (scope "materia" (many [route "plp" "ver_PLP", route "tleng" "ver_TLeng", route "bbdd" "ver_
         BBDD"])) ~=? ["materia/plp","materia/tleng","materia/bbdd"],
    paths (many [route "index" "ver_indice", scope "alumno" (route "materia/:nombre" "ver_materia")])
         ~=? ["index","alumno/materia/:nombre"],
    paths (many [route "index" "ver_indice", scope "alumno" (many [route "foo" "ver_foo", route ":lu/
         bar" "ver_bar"])]) ~=? ["index","alumno/foo","alumno/:lu/bar"],
    paths rutasFacultad ~=? ["","ayuda","materia/:nombre/alu/:lu/inscribir","materia/:nombre/alu/:lu/
         aprobar","alu/:lu/aprobadas"]
    ]


testsExecEntity = test [
    Just "root_url" ~=? exec path5 "",
    Just "post#index" ~=? exec path5 "post",
    Just "post#show" ~=? exec path5 "post/35",
    Just "category#create_of_7" ~=? exec path5 "category/7/create"
    ]

testsExec = test [
    exec (route "foo/bar/:lu" length) "foo/bar/001" ~=? Just (1),
    exec (scope "alu/:lu/materia/:nombre/:cuatri" (route "aprobada?" length) ) "alu/002/materia/plp/2
         c2015/aprobada?" ~=? Just(3),
    exec (many [route "index" (\ctx -> 0) , scope "alumno" (many [route "foo" (\ctx -> 1) , route ":lu/
         bar" (\ctx -> 2) ])]) "alumno/foo" ~=? Just(1),
    exec (route "foo/bar/:lu" length) "foo/bar/lu/001" ~=? Nothing,
    exec (scope "alu/:lu/materia/:nombre/:cuatri" (route "aprobada?" length) ) "alu/002/materia/plp/2
         c2015" ~=? Nothing,
    exec (many [route "index" (\ctx -> 0) , scope "alumno" (many [route "foo" (\ctx -> 1) , route ":lu/
         bar" (\ctx -> 2) ])]) "alumno/001/foo" ~=? Nothing
    ]

main = do
    runTestTT allTests
```