LOFT (Traductor de Lógico a Funcional)

LOFT es un paquete experimental diseñado para investigar si los programas escritos
En lenguaje lógico pueden ser traducido automáticamente a lenguaje funcional.

LOFT puede traducir un pequeño subconjunto del lenguaje lógico MERCURY al lenguaje funcional HASKELL.

Se espera que LOFT provea de un marco básico que pueda ser extendido para traducir
subconjuntos de MERCURY tan amplios como se requiera.

MERCURY

La entrada al traductor (LOFT) es un subconjunto de código fuente MERCURY.
El subconjunto de MERCURY que LOFT traduce actualmente se expone más adelante.
Para más información sobre MERCURY véase
Conway,Henderson & Somogyi, 1994.

Subconjunto

Prog::= (clause | mode | pred)
mode::= “:-” “mode” fn “(” modes “)” “is” det “.”
modes::= m (“,” m))*
det::= “det”
pred::= “:-” “pred” fn “(” modes “)” “.”
types::= p (“,” p)*
clause:= (fn “(” args “)” “.”
| fn “(” args “)” “:=” body “.”)
| fn “(” args “)” “:=” cond “.”)
args::= term (“,” term)*
body::= bodyitem (“,” bodyitem)*

bodyitem::= fn “(” args “)”
cond:= “(” guard “->” body “;” body “)”
guard::= term op term
const::= int | atom
m::= (“in” | “out”)
p::= (“int” | “list(int)”)
atom::= lowercase + (alphanummeric)*
var::= uppercase + (alphanummeric)*
int::= digit + (digit)*
op::= (“<” | “>” | “=<” | “>=” | “=” | “\=”)

La salida es código fuente Haskell.Para más información véase Hudak & Wadler,1991

EL PAQUETE LOFT

El paquete LOFT está escrito en Haskell y se puede usar interactivamente con un intérprete de Haskell como HUGS o bien compilado usando un compilador de Haskell
Como GHC.
El paquete traductor consta de 4 partes.
La primera el analizador lexicográfico , lee un fichero fuente en MERCURY y lo traduce a una lista de tokens simples.
Entonces el (parser) convierte esta lista de tokens en una estructura abstracta de datos.
A continuación el traductor transforma esta estructura en una forma conveniente para su impresión como programa funcional
Por último (pretty printer) imprime esa estructura en código fuente Haskell.

LEXER

El analizador lexicográfico es tomas código fuente MERCURY y separarlo en tokens para más adelante ser procesador por el (parser).

Estos tokens comprenden:

atom::= lowercase + (alphanummeric)*
var::= uppercase + (alphanummeric)*
int::= digit + (digit)*
op::= (“<” | “>” | “=<” | “>=” | “=” | “\=”)
symbol::= dicersor símbolos p.ej: “:-” “mode”

PARSER

El propósito del (parser) es tomar una lista de tokens producida por el analizador lexicográfico y convertirla en una estructura abstracta de datos representando un programa MERCURY.

prog::= (clause | mode | pred)*
mode::= functor + modes + det
det::= “det”
pred::= functor + types
clause::= functor + args + body
args::= (term)*
body::= (bodyitem)* | cond | empty
bodyitem::= functor + args
cond::= guard + body + body
guard::= term + op + term
functor::= atom
term::= (const | var)
const::= (int | atom)

Esto se hace usando un (parser) descendente recursivo implementado usando (combinators) como los descritos en Wadler,1985.

TRASLATOR

El traductor se basa en reglas de reescritura implementadas en Haskell. Hay muy poca diferencia entre las reglas de reescritura y código Haskell.
La principal diferencia es que la notación conjunto utilizada en las reglas de reescritura tiene que ser implementada usando listas Haskell.

PRETTY PRINTER

Pretty printer es código Haskell de varias líneas simples que toma una estructura cuasiHaskell producida por el traductor e imprime la salida en código fuente Haskell.

EJEMPLO:

Partición

```
:- pred part(list(int),int,list(int),list(int)).
:- mode part(in,in,out,out) is det.
part([X|Xs],Y,Ps,Qs):-
   (X=<Y -> part(Xs,Y,Ls,Bs),Ps=[X|Ls],Qs=Bs
         ; part(Xs,Y,Ls,Bs),Ps=Ls,Qs=[X|Bs]).
part([],X,[],[]).
```

```
part::[Int]->Int->([Int],[Int])
part (x:xs) y | x <= y = (ps,qs)
   where (ls,bs) = part xs y
         ps = (x:ls)
         qs = bs
```

```
part (x:xs) y | otherwise = (ps,qs)
   where (ls,bs) = part xs y
         ps = ls
         qs = (x:bs)
part [] x = ([],[])
```

Quicksort

```
:- pred qs(list(int),list(int)).
:- mode qs(in,out) is det.
qs([X|Xs],Ys):-part(Xs,X,Littles,Bigs),
               qs(Littles,Ls),
               qs(Bigs,Bs),
               app(Ls,[X|Bs],Ys).
qs([],[]).
```

```
qs::[Int]->[Int]
qs (x:xs) = ys
   where (littles,bigs) = part xs
         ls = qs littles
         bs = qs bigs
         ys = app ls (x:bs)
qs [] = []
```

Prefix

```
:- pred prefix(list(int),list(int)).
:- mode prefix(in,out) is nondet.
prefix([X|Xs],[X|Ys]):-prefix(X,Ys).
prefix([],Y).
```

LOFT podría traducir prefix si fuera determinista sin embargo no sirve cuando es determinista y no operará cuando sea no determinista.

Suffix

```
:- pred suffix(list(int),list(int)).
:- mode suffix(in,out) is nondet.
suffix(Xs,[Y|Ys]:-suffix(Xs,Ys).
suffix(Xs,Xs).
```

Suffix tiene los mismos problemas que prefix

Reverse

```
:- pred reverse(list(int),list(int)).
:- mode reverse(in,out) is det.
reverse([X|Xs],Zs):-reverse(Xs,Ys),append(Ys,[X],Zs).
reverse([],[]).
```

```
reverse::[Int]->[Int]
reverse (x:xs) = zs
   where
   ys = reverse xs
   zs = append ys [x]
```

RESUMEN

El paquete LOFT traduce un subconjunto de programas en MERCURY en programas Haskell.
Está escrito (en sí mismo) en Haskell y está basado en un (combinator pattern matcher).
El paquete está compuesto de 4 partes : lexer, parser ,translator & pretty printer.