Paradigmas de Programación
FAMAF - UNC 2019

Notas sobre el Lab 1: Programación Funcional

Deep vs. Shallow embedding

Una primera intuición nos llevó a querer definir el EDSL como *shallow-embedded*. Quisimos traducir las definiciones del lenguaje a funciones de *gloss*:

```
Rotar a = rotate 90 a
```

> *Si lo hubiéramos hecho así, habríamos simplemente* traducido *gloss* al español.

Descifrando la función `initial` y releyendo las consignas comprendimos que nuestro EDSL debía ser *deeply-embedded*. `Dibujo` debía ser un tipo de datos:

```
data Dibujo a = Simple a | Rotar (Dibujo a)| Espejar (Dibujo a) | Rot45 (Dibujo a)
              | Apilar Int Int (Dibujo a) (Dibujo a)
              | Juntar Int Int (Dibujo a) (Dibujo a)
              | Encimar (Dibujo a) (Dibujo a)
```

> *Usamos* `Simple a` *en vez de* `Basica a` *porque la palabra* `Basica` *ya estaba definida. Queríamos evitar confusiones.*

Traducir figuras vs. interpretar el lenguaje

Nos tomó un tiempo comprender la diferencia entre `interpBas` e `interp`. Nuestra primera intuición fue querer definir el intérprete del lenguaje en `interpBas`. No funcionó.

Entendemos a `interpBas` como un *diccionario* que dice cómo traducir las figuras básicas a `FloatingPic`'s. El intérprete del lenguaje se define en `interp`.

Interpretando transformaciones

Se pueden usar las transformaciones nativas de `gloss`.

```
interp f (Rotar d) a b c = rotate 90 (interp f d a b c)
```

Sin embargo, ¡así no se usa la semántica del lenguaje definida en las consignas! Es necesario modificar explícitamente los parámetros *a*, *b* y *c*.

```
interp f (Rotar d) a b c = fig (a V.+ b) c (opposite b)
                           where fig = interp f d
```

> `fig` *es un* `FloatingPic`*; o sea,* `trian1` *o* `fShape` *o cualquier otro.*

> *Usamos la función auxiliar* `opposite = (zero V.-)`.

Unión de figuras

En la semántica hay operaciones definidas como unión de figuras. Usamos `pictures [a]` (nativo de *gloss*) para implementar la unión.

Creamos la función auxiliar `union` que permite *abstraer a* gloss *del intérprete.*

```
union :: Picture -> Picture -> Picture
union p1 p2 = pictures [p1, p2]
```

> *Ahora, al reemplazar a* gloss *por otra librería no hay que cambiar la definición de* `interp`.

Functores, aplicativos y mónadas

Descubrimos que la función `cambia` es simplemente una versión de *bind* (`>>=`) con los argumentos al revés. (En efecto, `(>>=) = flip cambia`.) *Bind* es una función de la clase `Monad`.

Las mónadas son subclases de los aplicativos, que son subclases de los functores. Teníamos ya definidas las funciones `fmap` y `pure`. Se nos presentó el desafío de definir `<*>`.

Observando cómo se comporta el operador `<*>` con arreglos, decidimos qué sentido darle a `d1 <*> d2`.

> *Tener en cuenta que* $d1$ *es un árbol cuyas hojas son funciones.*

El resultado de la aplicación de $d1$ en $d2$ es un árbol construido a partir de $d1$, donde **se reemplaza cada hoja de** $d1$ **por un árbol con estructura idéntica a** $d2$. Las *funciones-hoja* de $d1$ se usan para *mapear* al sub-árbol que las va a reemplazar.

> *El operador* `<*>` *cumple con la propiedad* `Simple f <*> d = mapDib f d`.

Esto resultó ser increíblemente poderoso. Por ejemplo, usando la *composición n-veces* y la función `cuarteto1` (que repite una figura en cuatro cuadrantes) se puede hacer una estructura fractal (aunque no infinita).

```
cuartetoId = cuarteto1 $ Simple id
ejemplo = (comp (cuartetoId <*>) 6) $ Simple Triang
```

Por otro lado, entendimos que `mapDib` es un caso particular de `cambia`. En efecto, toda mónada es también un functor.

```
mapDib f d = cambia (Simple . f) d
```

Semántica

Algunas funciones definidas en `Predicados.hs` utilizan la función `sem`.

`sem` funciona parecido a `foldl` en listas: *reduce* el árbol a un solo valor. La diferencia es que en vez de tomar una sola función reductora, toma *una función para cada caso del pattern-matching*. Es decir, al usar `sem` estamos haciendo **pattern-matching indirecto**, donde **el orden** de las funciones pasadas a `sem` indica **cómo manejar cada caso**.

Escher

Las funciones en este archivo se definieron siguiendo exactamente los dibujos y las fórmulas en el paper de Henderson. (¡Igual tuvimos que pensarlo mucho!)