

# Functional Reactive Programming

Gonzalo Castiglione, Juan Pablo Rey

February 16, 2014

## Abstract

Para el presente trabajo se decidió aplicar Arrow-based Functional Reactive Programming [1] junto a Elm para realizar una simulación de un “ant colony”.

El objetivo consiste en aplicar lo aprendido en la materia Programación Funcional junto con AFRP para realizar una demo en donde hormigas recorren un tablero en busca de comida para luego traerla al hormiguero y guardarla.

## Aplicación

Para la simulación se consideran las siguientes entidades:

- Comida: Simplemente existe en algunos lugares del mapa.
- Hormigas: Recorren el tablero en busca de comida.
- Hormiguero: Representa un depósito de comida de las hormigas.
- Piedras: Obstáculos que existen en el mapa que no pueden ser traspasados ni movidos.
- Feromona: Sustancia que las hormigas dejan en su camino al encontrar comida y volver al hormiguero.

El funcionamiento de la aplicación funciona de la siguiente forma:

Las hormigas recorren el mapa en búsqueda de comida. Cuando alguna encuentra comida, esta carga tanto como pueda (deja el resto en el lugar) y la lleva directamente al hormiguero para almacenarla. Durante su regreso al hormiguero, se libera feromona en cada paso. Cuando una hormiga cruza un rastro de feromona, esta sigue con preferencia el lado que mas feromona tenga. Esto producirá que las fuentes de comida, atraigan con facilidad a muchas hormigas.

La feromona tiene una función de decaimiento dependiente del tiempo. Es decir, la feromona que fue liberada solamente seguirá en el lugar durante un cierto tiempo solamente.

El algoritmo hormiga explicado en detalle escapa a los objetivos del proyecto pero puede encontrarse en [2].

## Obstáculos

Debido a que Elm se encuentra todavía en etapa de desarrollo. Aún no están soportadas las type classes [3]. Por lo que las arrows no están soportadas, simplemente existen primitivas que permiten su implementación. Se optó entonces por realizar la implementación de las siguientes:

- Arr

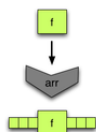


Figure 1: Arr

- Composición inversa - >>>

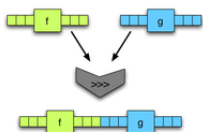


Figure 2: Composición Inversa

- fork - &&&

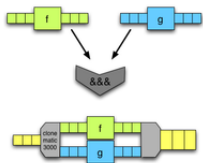


Figure 3: And

- or - \*\*\*

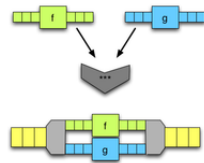


Figure 4: Or

- first

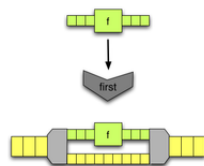


Figure 5: First

- Otros

**identity:** Simplemente retorna en su output lo mismo que ingresó como input

**composición** ( $<<<$ ): Es análoga a la composición inversa, con el orden de aplicación intercambiado

**second:** Realiza lo mismo que first, con la diferencia que alimenta la segunda parte de cada par de entrada con el arrow indicado en vez del primero

## Referencias

- [1] - AFRP: [http://www.haskell.org/haskellwiki/Functional\\_Reactive\\_Programming](http://www.haskell.org/haskellwiki/Functional_Reactive_Programming)
- [2] - Algoritmo hormiga: [http://en.wikipedia.org/wiki/Ant\\_colony\\_optimization\\_algorithms](http://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms)
- [3] - Type classes: [http://en.wikipedia.org/wiki/Type\\_class](http://en.wikipedia.org/wiki/Type_class)