Tutorial on ReactiveML
Sept. 2018

ICFP, Saint-Louis, USA
G. Baudart, L. Mandel,
C. Pasteur, & M. Pouzet

The Darwin sieve in ReactiveML

The goal of this exercise is to program the Darwin sieve in ReactiveML. This example has been presented by Gérard Berry at Collège de France (“leçon inaugurale”, 11/19/2009; lesson no. 5, 01/06/2010 and lesson no. 8, 01/27/2010) to illustrate the “chemical abstract machine”, a model of concurrency which describe computations as chemical reaction rules.

The principle of the Darwin sieve is to represent a set of integer numbers as a collection of processes. Each process can move in any direction (here in 2D space). When two numbers are in contact, if the former divides the later, the later disappear. After some time, only prime numbers stay alive! [1]

The skeleton of the ReactiveML program is available here:

`http://reactiveml.org/icfp18/darwin.tgz`

Remark. To compile an ReactiveML file `file.rml`, first generate OCaml code by typing:

```
  --> rmlc file.rml
```

Then, compile the generated OCaml file and link it to the `unix.cma` and `rmllib.cma` libraries to create an executable:

```
  --> ocamlc -o prog -I ‘rmlc -where‘ unix.cma rmllib.cma file.ml
```

1 Moving numbers

Each number is represented by a small disk that moves in a 2D space, starting with an initial position and speed. Its behavior is to goes straight until it elastically bounces on a wall. To represent a number and the walls, you can use for exmple the following data-structures:

```
type coord = { x: float; y: float; }

type number_state =
    { id: int;
      pos: coord;
      speed: coord;
```

[1] A fun illustration was given by Gérard Berry using a fish tank. The fish number 2 eats all the even fishes, the fish number 3 eats fishes that are multiple of 3, and so on!

```
      radius: float;
      color: Graphics.color; }

type wall = { left: float; right: float;
              bot: float; top: float }
```

Question 1
Define a process:

```
  moving_number:
    number_state -> wall -> (number_state, 'a) event -> unit process
```

so that `run (moving_number init_state s)` *moves a number whose initial state is* `init_state`*. At every instant, the process must emit the current state on the signal* `s`*.*

Question 2
Define a process `window: wall -> ('a, number_state list) event -> unit process` *which allows for observing the Darwin sieve.* `run (window wall s)` *initializes the graphics mode then, at every instant, it receives on the signal* `s` *the state of all numbers involved in the sieve and displays those numbers.*

You can use the function `draw_number: number_state -> unit` *which displays a number on the graphical window.*

Question 3
Write a function `random_number_state: int -> wall -> number_state` *which create a value of type* `number_state` *such that the field* `id` *is equal to the integer given as the first argument, the fields* `pos` *and* `speed` *are initialized with a random value, the field* `radius` *is equal to* `12.0` *and* `color` *is equal to* `Graphics.cyan`*.*

Question 4
Write a process `main: unit process` *which execute in parallel one hundred instances of the process* `moving_number` *and one instance of the process* `window`*.*

You can use the construct `for/dopar` *of ReactiveML.*

2 Collisions

We deal now with the removal of a number when it collides with a number that divides it. For that, with associate a signal `kill` to every number:

```
type number_state =
    { id: int;
      pos: coord;
      speed: coord;
      radius: float;
      color: Graphics.color;
      kill: (number_state, number_state option) event; }
```

Question 5
Modify your program to incorporate the change of the data-type `number_state`*.*

Question 6
Define a process

```
number: number_state -> wall -> (number_state, 'a) event -> unit process
```

so that `run (number init_state wall s)` *moves the number in initial position* `init_state` *with a behavior defined by the process* `moving_number` *until its associated signal* `kill` *is emited. Once the signal* `kill` *is received, the number becomes red and its size must progressively reduce. When the radius is nul, the processes terminates.*

Question 7
Manage the collisions so that numbers that are not prime numbers are removed.

3 Dynamic creation

The goal of this part is to dynamically create numbers every tim the user clicks on the mouse button.

Question 8
Define a reactive process:

```
add: ('a, coord) event ->
       wall -> int -> (number_state, 'b) event -> unit process
```

such that `add new_number wall n s` *creates a new number, starting from* `n`*, every time it receive a position on the signal* `new_number`*.*

Question 9
Consider the following process `click_of_button_down: (coord, 'a) event -> unit process` *which emits the current position of the mouse every time the mouse button is pressed:*

```
let process click_of_button_down click =
  loop
    if Graphics.button_down() then begin
      let x, y = Graphics.mouse_pos() in
      emit click { x = float_of_int x; y = float_of_int y}
    end;
    pause
  end
```

Write a process `read_click: (coord, 'a) event -> unit` *such that* `read_click new_number` *emits on the signal* `new_number` *the coordinates of the mouse every time the mouse button is released (that is, on the falling edge of the sequence of clicks).*

Question 10
Rewrite the process `main` *so that it manages the dynamic creation of numbers.*