Paradigmas de Programación

Práctica 5

La conjetura de Collatz

Nota Importante:

Cuando se solicite la entrega de esta práctica, cada alumno deberá subir a su repositorio de prácticas (del cual se indicará su ubicación más adelante) un directorio `p5` cuyo contenido debe ser únicamente el fichero `collatz.ml`.

Sea muy cuidadoso a la hora de crear el directorio y los ficheros, y **respete los nombres indicados**. En particular, fíjese que todos estos nombres sólo contienen letras en minúsculas, números y puntos.

Además, **todos los ficheros deben compilar sin errores** con las siguientes órdenes:

```
ocamlc -c collatz.ml
```

Ejercicios:

1. Considere la función f de $\mathbb{N} \to \mathbb{N}$, que podría aproximarse en OCaml con la siguiente definición para `f: int -> int`

   ```
   let f n = if n mod 2 = 0 then n / 2 else 3 * n + 1
   ```

   Según la Conjetura de Collatz[1], si partimos de cualquier número positivo y vamos aplicando repetidamente esta función, seguiremos un camino que llegará inexorablemente al 1. Por ejemplo, partiendo del 13, tendríamos el siguiente camino:

   $$13, 40, 20, 10, 5, 16, 8, 4, 2, 1$$

   Llamaremos "órbita" de un número al camino que se sigue de esta manera desde ese número hasta el 1 (ambos incluidos). Por ejemplo, la que hemos escrito más arriba es la órbita del 13.

   En un archivo con nombre `collatz.ml`, defina (de modo recursivo) en OCaml una "función" `orbit: int -> unit` tal que, cuando se aplique esta función a cualquier $n > 0$ se envíe a la salida estándar una línea con la órbita de n, siguiendo el formato del ejemplo (los distintos valores por los que pasa la órbita deben estar separados por una coma y un espacio y debe terminarse con un salto de línea).

   ```
   # orbit;;
   - : int -> unit = <fun>

   # orbit 13;;
   13, 40, 20, 10, 5, 16, 8, 4, 2, 1
   - : unit = ()

   # orbit 1;;
   1
   - : unit = ()
   ```

[1] https://es.wikipedia.org/wiki/Conjetura_de_Collatz

Añada al mismo archivo la definición (recursiva) de una función `length: int -> int`, tal que, para cualquier $n > 0$, `length n` sea el número de pasos necesarios (en la órbita de n) para llegar hasta el 1. Así, por ejemplo, `length 13` debe ser `9`.

```
# length;;
- : int -> int = <fun>

# length 13;;
- : int = 9

# length 27;;
- : int = 111
```

Añada al mismo archivo la definición (recursiva) de una función `top: int -> int`, tal que, para cada $n > 0$, `top n` sea el valor más alto alcanzado en la órbita de n. Así, por ejemplo, `top 13` debe ser `40`.

```
# top;;
- : int -> int = <fun>

# top 13;;
- : int = 40

# top 27;;
- : int = 9232
```

Defina también directamente (de modo recursivo, sin usar las funciones `length` y `top`) una función `length'n'top: int -> int * int`, tal que, para cada entero n, devuelva un par de enteros indicando la longitud de su órbita y su altura máxima. Así, por ejemplo, `length'n'top 13` debería ser el par `(9, 40)`. Se trata de que al aplicar esta definición "no se recorra dos veces la órbita en cuestión". (Añada esta definición al archivo `collatz.ml`).

```
# length'n'top;;
- : int -> int * int = <fun>

# length'n'top 13;;
- : int * int = (9, 40)

# length'n'top 27;;
- : int * int = (111, 9232)
```

2. **Ejercicio opcional**. Defina una función `longest_in: int -> int -> int` tal que `longest_in m n` devuelva el menor valor del intervalo $[m, n]$ cuya órbita tenga longitud maximal en ese intervalo. Así, por ejemplo, `longest_in 86 87` debería ser `86`. (Añada esta definición al archivo `collatz.ml`).

   Defina una función `highest_in: int -> int -> int` tal que `highest_in m n` devuelva el menor valor del intervalo $[m, n]$ cuya órbita tenga altura maximal en ese intervalo. Así, por ejemplo, `highest_in 86 87` debería ser `87`. (Añada esta definición al archivo `collatz.ml`).

```
# longest_in;;
- : int -> int -> int = <fun>

# highest_in;;
- : int -> int -> int = <fun>

# longest_in 1 1000;;
- : int = 871

# highest_in 1 1000;;
- : int = 703
```