Walter Cazzola

Home Page
ADAPT Lab.
Curriculum Vitae
Research Topic

Exam of Programming Languages

26 February 2015

Exercise OCaML/ML: Playing with Intervals.

```
open Comparable ;;
open IntervalI ;;

module Interval(Endpoint: Comparable):
     (IntervalI with type endpoint = Endpoint.t) =
  struct
    type endpoint = Endpoint.t
    type interval = Interval of Endpoint.t * Endpoint.t | Empty
    exception WrongInterval

    (** [create low high] creates a new interval from [low] to
        [high].  If [low > high], then the interval is empty *)
    let create low high =
      if Endpoint.compare low high > 0 then raise WrongInterval
      else if Endpoint.compare low high == 0 then Empty
      else Interval (low,high)

    (** Returns true iff the interval is empty *)
    let is_empty = function
      | Empty -> true
      | Interval _ -> false

    (** [contains t x] returns true iff [x] is contained in the
        interval [t] *)
    let contains t x =
      match t with
      | Empty -> false
      | Interval (l,h) ->
        Endpoint.compare x l >= 0 && Endpoint.compare x h <= 0

    (** [intersect t1 t2] returns the intersection of the two input
        intervals *)
    let intersect t1 t2 =
      let min x y = if Endpoint.compare x y <= 0 then x else y in
      let max x y = if Endpoint.compare x y >= 0 then x else y in
      match t1,t2 with
      | Empty, _ | _, Empty -> Empty
      | Interval (l1,h1), Interval (l2,h2) ->
        create (max l1 l2) (min h1 h2)

    let tostring = function
    | Empty -> "[]"
    | Interval(l,h) -> "["^(Endpoint.tostring l)^", "^(Endpoint.tostring h)^"]"
  end ;;

module IntInterval = Interval(
  struct
    type t = int
    let compare = compare
    and tostring = string_of_int
  end);;
module StringInterval = Interval(
  struct
    type t = string
    let compare = String.compare
    and tostring = fun x -> x
  end);;
```

Exercise Erlang: Distributed Tasks.

```
-module(ring).
-export([start/2, stop/0, send_message/1, send_message/2, create/3]).

start(NumberProcesses, F) ->
  register(
     ring_recursion_functional,
```

```
    spawn(?MODULE, create, [NumberProcesses, F, self()])),
  receive
    ready      -> ok
    after 5000 -> {error, timeout}
  end.

stop() -> ring_recursion_functional ! {command, stop}.

send_message(Message) -> send_message(Message, 1).
send_message(Message, Times) ->
  ring_recursion_functional ! {command, message, [Message, Times]}.

create(1, [H|_], Starter) ->
  Starter ! ready,
  loop_last(ring_recursion_functional, H);
create(NumberProcesses, [H|TL], Starter) ->
  Next = spawn_link(?MODULE, create, [NumberProcesses-1, TL, Starter]),
  loop(Next, H).


loop_last(NextProcess, F) ->
  receive
    {command, stop} ->  exit(normal),unregister(ring_recursion_functional);
    {command, message, [Message, 1]} ->
        io:format("~p~n", [F(Message)]),
        loop_last(NextProcess, F);
    {command, message, [Message, Times]} ->
        NextProcess ! {command, message, [F(Message), Times-1]},
        loop_last(NextProcess, F)
  end.

loop(NextProcess, F) ->
  receive
    Msg = {command, stop} ->
        NextProcess ! Msg,
        ok;
    {command, message, [Message, Times]} ->
        NextProcess ! {command, message, [F(Message), Times]},
        loop(NextProcess, F)
  end.
```

On Your Desk.

```
import scala.util.parsing.combinator._
import scala.collection.mutable._

class DeskCombinators(var the_table: HashMap[Char,Int])
         extends JavaTokenParsers {
   def program = "print" ~> expr ~
        ( "where" ~> rep1sep(decl, ",")) ^^ {
            case f ~ d => println(f()); the_table }
   def expr: Parser[() => Int] =
     (
        (( (num~("+" ~> expr))
     | (x ~ ("+" ~> expr))
     ) ^^ {case f1~f2 => () => f1() + f2()}:Parser[()=>Int])| x | num)
   def x = """[a-z]""".r ^^ {c => () => the_table(c.charAt(0))}
   def num = wholeNumber ^^ { n => () => n.toInt }

   def decl = """[a-z]""".r ~ ("=" ~> wholeNumber) ^^ {
       case c~n => the_table(c.charAt(0)) = n.toInt}
}

object DeskEvaluator {
  def main(args: Array[String]) = {
     val p = new DeskCombinators(new HashMap[Char,Int]())

     args.foreach { filename =>
        val src = scala.io.Source.fromFile(filename)
        val lines = src.mkString
        p.parseAll(p.program, lines) match {
            case p.Success(s,_) => println(s)
            case x => print(x.toString)
        }
        src.close()
     }
  }
}
```