A SMALL ARITHMETIC INTERPRETER (45/45 points)

In this exercise, we will write a small program that computes some operations on integers. We will use a small datatype `operation` that describes all the operations to perform to compute the result. For example, suppose we want to do the following computation:
`mul (add 0 1) (mul 3 4)`
We can describe it as:
`Op ("mul", Op ("add", Value 0, Value 1), Op ("mul", Value 3, Value 4))`
The `Op` constructor takes as a first argument a `string`, which is the name of the function that is stored in an `environment`. We suppose there exists a variable `initial_env` that contains some predefined functions.

1. First of all, we need a way to find a function in an environment of type `env`, which is basically a list of tuples. Each of these tuples contains a `string`, which is the name of the function, and a value of type `int -> int -> int`, which is basically a function that takes two arguments of type `int` and returns an `int` as a result.
   Write a function `lookup_function : string -> env -> (int -> int -> int)` that returns the function associated to a name in an environment. If there is no function with the name given, you can return `invalid_arg "lookup_function"`.
2. Another useful feature would be to add functions to a given environment. Write the function `add_function : string -> (int -> int -> int) -> env -> env` that takes an environment `e`, a name for the function `n` and a function `f`, and returns a new environment that contains the function `f` that is associated to the name `n`.

   What you can notice now is that unless you put explicit annotations, those two previous functions should be polymorphic and work on any list of couples. Actually, `lookup_function` could have been written as `List.assoc`.
3. Create a variable `my_env: env` that is the initial environment plus a function associated to the name `"min"` that takes two numbers and returns the lowest. You cannot use the already defined `Pervasives.min` function, nor any `let .. in`. Take advantage of lambda expressions!
4. Now that we have correctly defined the operations to use the environment, we can write the function that computes an operation. Write a function `compute: env -> operation -> int` that takes an environment and an operation description, and computes this operation. The result is either:
   - Directly the value.
   - An operation that takes two computed values and a function from the environment.
5. Let's be a bit more efficient and use the *over-application*: suppose a function `id: 'a -> 'a`, then `id id` will also have type `'a -> 'a`, since the `'a` is instantiated with `'a -> 'a`. Using that principle, we can apply `id` to itself infinitely since it will always return a function. Write a function `compute_eff : env -> operation -> int` that takes an environment and an operation, and computes it. However, you cannot use `let` inside the function!

THE GIVEN PRELUDE

```
type operation =
    Op of string * operation * operation
  | Value of int

type env = (string * (int -> int -> int)) list
```

YOUR OCAML ENVIRONMENT

```
let add_function name op env =
  (name, op) :: env
;;

let my_env =
  add_function "min" (fun a b -> if a < b then a else b) initial_env
;;


let rec compute env op = match op with
  | Value e -> e
  | Op (name, op1, op2) ->  (lookup_function name env) (compute env op1) (compute env op2)
;;

let rec compute_eff env = function
  | Value e -> e
  | Op (name, op1, op2) ->  (lookup_function name env) (compute env op1) (compute env op2)
;;
```

Switch >>

Typechecl

Reset Templ

Full-screen |

Check & Sa

Exercise incomplete (click for details) 45 pts

v Exercise 1: lookup_function Completed, 10 pts

Found a toplevel definition for `lookup_function` .

Found `lookup_function` with compatible type.

```
Computing
lookup_function "sub" [("xor", xor); ("mod", mod); ("or", or); ("and", and)]
Correct exception Invalid_argument(lookup_function)                     1 pt

Computing
  lookup_function
    "and"
    [("add", add); ("mul", mul); ("xor", xor); ("and", and); ("mod", mod)]
Correct value and                                                       1 pt

Computing lookup_function "or" [("sub", sub)]
Correct exception Invalid_argument(lookup_function)                     1 pt

Computing
  lookup_function
    "or"
    [("or", or); ("div", div); ("sub", sub); ("and", and); ("xor", xor);
     ("mul", mul); ("add", add)]
Correct value or                                                        1 pt

Computing
  lookup_function
    "div"
    [("and", and); ("sub", sub); ("xor", xor); ("mul", mul); ("add", add)]
Correct exception Invalid_argument(lookup_function)                     1 pt

Computing
  lookup_function
    "add"
    [("mul", mul); ("add", add); ("mod", mod); ("xor", xor); ("or", or)]
Correct value add                                                       1 pt

Computing lookup_function "and" [("or", or); ("and", and); ("mod", mod)]
Correct value and                                                       1 pt

Computing
lookup_function "xor" [("or", or); ("div", div); ("and", and); ("add", add)]
Correct exception Invalid_argument(lookup_function)                     1 pt

Computing
  lookup_function
    "add"
    [("mul", mul); ("or", or); ("add", add); ("mod", mod); ("xor", xor);
     ("div", div); ("and", and)]
Correct value add                                                       1 pt

Computing
  lookup_function
    "div"
    [("xor", xor); ("mod", mod); ("and", and); ("add", add); ("or", or)]
Correct exception Invalid_argument(lookup_function)                     1 pt
```

v Exercise 2: add_function Completed, 10 pts

Found `add_function` with compatible type.

```
Computing
  add_function
    "mul"
    mul
```

```
     ("xor", xor)]
Computing
  add_function
    "or"
    or
    [("xor", xor); ("sub", sub); ("mod", mod); ("and", and); ("mul", mul);
     ("div", div)]
Correct value                                                              1 pt
  [("or", or); ("xor", xor); ("sub", sub); ("mod", mod); ("and", and);
   ("mul", mul); ("div", div)]
Computing add_function "or" or [("sub", sub); ("xor", xor); ("mod", mod)]
Correct value [("or", or); ("sub", sub); ("xor", xor); ("mod", mod)]       1 pt
Computing
  add_function
    "mul"
    mul
    [("xor", xor); ("div", div); ("add", add); ("and", and)]
Correct value [("mul", mul); ("xor", xor); ("div", div); ("add", add); ("and", and)]1 pt
Computing add_function "and" and [("div", div); ("xor", xor); ("mod", mod)]
Correct value [("and", and); ("div", div); ("xor", xor); ("mod", mod)]     1 pt
Computing
  add_function
    "add"
    add
    [("sub", sub); ("mul", mul); ("or", or); ("mod", mod); ("div", div)]
Correct value                                                              1 pt
  [("add", add); ("sub", sub); ("mul", mul); ("or", or); ("mod", mod);
   ("div", div)]
Computing add_function "div" div [("sub", sub)]
Correct value [("div", div); ("sub", sub)]                                 1 pt
Computing
  add_function
    "mul"
    mul
    [("mod", mod); ("and", and); ("add", add); ("or", or); ("sub", sub);
     ("div", div)]
Correct value                                                              1 pt
  [("mul", mul); ("mod", mod); ("and", and); ("add", add); ("or", or);
   ("sub", sub); ("div", div)]
Computing add_function "sub" sub [("add", add)]
Correct value [("sub", sub); ("add", add)]                                 1 pt
Computing add_function "mod" mod []
Correct value [("mod", mod)]                                               1 pt
```

v Exercise 3: my_env — Completed, 5 pts

```
Found a toplevel definition for my_env .
Found my_env with compatible type.
The min function has correctly been added to my_env
Computing min 9 -19
Correct value -19                                                          1 pt
Computing min 5 -17
Correct value -17                                                          1 pt
Computing min 24 -19
Correct value -19                                                          1 pt
Computing min -10 17
Correct value -10                                                          1 pt
Computing min -2 23
Correct value -2                                                           1 pt
```

v Exercise 4: compute — Completed, 10 pts

```
Found compute with compatible type.
Computing
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("mul", Op ("mul", Value (-3), Value (-2)),
      Op ("sub", Value 2, Op ("or", Value 0, Value 0))))
Correct value 12                                                           1 pt
Computing
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("sub", Value 2,
      Op ("sub",
       Op ("mul", Value (-4),
        Op ("or", Value (-1), Op ("sub", Value 3, Op ("or", Value 2, Value 4)))),
       Value 0)))
Correct value -2                                                           1 pt
Computing
```

```
        Op ("mul", Op ("or", Value 3, Value (-1)), Value (-1)))
```

Correct value 0 — 1 pt

Computing
```
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("sub",
      Op ("or", Value (-2),
       Op ("or", Value 2,
        Op ("sub", Value 2,
         Op ("or", Value 0,
          Op ("sub", Value 0,
           Op ("mul", Value (-1), Op ("or", Value 2, Value 2))))))),
      Value (-1)))
```
Correct value -1 — 1 pt

Computing `compute [("or", or); ("sub", sub); ("mul", mul)] (Value 0)`

Correct value 0 — 1 pt

Computing
```
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("mul", Value 2, Op ("mul", Op ("sub", Value 3, Value (-3)), Value 3)))
```
Correct value 36 — 1 pt

Computing
```
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("or", Op ("sub", Value (-1), Value 1), Value (-5)))
```
Correct value -1 — 1 pt

Computing
```
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("or", Value 2, Value 1))
```
Correct value 3 — 1 pt

Computing
```
  compute
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("mul", Value (-5),
      Op ("mul",
       Op ("or",
        Op ("or", Value 1,
         Op ("sub", Value (-1),
          Op ("sub", Op ("mul", Value (-4), Op ("mul", Value 1, Value (-1))),
           Value 4))),
        Value (-2)),
       Value 3)))
```
Correct value 15 — 1 pt

Computing `compute [("or", or); ("sub", sub); ("mul", mul)] (Value (-5))`

Correct value -5 — 1 pt

v Exercise 5: `compute_eff` — Incomplete, 10 pts

Found a toplevel definition for `compute_eff`.

You cannot reuse the compute function. — 0 pt

You cannot reuse the compute function. — 0 pt

Found `compute_eff` with compatible type.

Computing `compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value (-1))`

Correct value -1 — 1 pt

Computing
```
  compute_eff
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("sub", Value 3,
      Op ("or",
       Op ("mul",
        Op ("or", Value 0,
         Op ("mul", Op ("sub", Value (-4), Op ("sub", Value 3, Value 0)),
          Value 4)),
        Value (-2)),
       Value (-4))))
```
Correct value 7 — 1 pt

Computing `compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value (-1))`

Correct value -1 — 1 pt

Computing `compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value (-4))`

Correct value -4 — 1 pt

Computing
```
  compute_eff
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("or", Value 0, Value (-1)))
```
Correct value -1 — 1 pt

Computing `compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value (-1))`

Correct value -1 — 1 pt

Computing `compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value (-2))`

```
Computing compute_eff [("or", or); ("sub", sub); ("mul", mul)] (Value 1)
Correct value 1                                                                1 pt
Computing
  compute_eff
    [("or", or); ("sub", sub); ("mul", mul)]
    (Op ("mul", Op ("sub", Value (-1), Value 3), Value 4))
Correct value -16                                                              1 pt
```