```
(*
                              CS51 Lab 9
                        Substitution Semantics

   Objective:

   In this lab, you'll gain practice with understanding and generating
   substitution semantic derivations, along with the formal
   definitions of free variables and substitution.
 *)

(*======================================================================
Part 1: Substitution semantics derivation

In this part of the lab, you'll work out the formal derivation of the
substitution semantics for the expression

    let x = 3 + 5 in
    (fun x -> x * x) (x - 2)

according to the semantic rules presented in Chapter 13.

        ******************************************
        FOR THIS LAB YOU WILL NEED PENCIL AND PAPER
        OR SOME SIMILAR WRITING SYSTEM.
        ******************************************

Before beginning, what should this expression evaluate to? Test out
your prediction in the OCaml REPL. *)

(* The exercises will take you through the derivation stepwise, so
that you can use the results from earlier exercises in the later
exercises.

By way of example, we do the first couple of exercises for you to give
you the idea.

    *****************************************************************
    Note: When we ask for a derivation using a particular set of rules
    -- such as the substitution rules in Figure 13.4 or the semantic
    rules in Figure 13.5 -- the derivation should be explicit about
    that particular set of rules, but should not explicitly show the
    use of any other rules. For instance, in Exercise 3, the solution
    shows the semantic rules (Figure 13.5) in the derivation
    explicitly. Implicitly, it's making use of the substitution rules
    (Figure 13.4) by its appeal to Exercise 2.
    *****************************************************************

......................................................................
Exercise 1. Carry out the derivation for the semantics of the
expression `3 + 5`, using the semantics rules in Figure 13.5.
....................................................................*)

(* SOLUTION:

    3 + 5 â\207\223
          | 3 â\207\223 3          (R_int)
          | 5 â\207\223 5          (R_int)
          â\207\223 8              (R_+)

   (We try to mimic the notation for semantic rules and derivations
   from the textbook, but the appearance is imperfect.)

   This derivation was actually given in the reading in Section
   13.1. We've annotated each line with the semantic rule that it
```

```
   uses. You should do that too below.
 *)

(*......................................................................
Exercise 2. What is the result of the following substitution according
to the definition in Figure 13.4?

    (x + 5) [x â\206¦ 3]
......................................................................*)

(* SOLUTION: Carrying out each step in the derivation:

    (x + 5) [x â\206¦ 3]
        = x [x â\206¦ 3] + 5 [x â\206¦ 3]      (by 13.10)
        = 3 + 5 [x â\206¦ 3]              (by 13.8)
        = 3 + 5                      (by 13.7)

   Again, we've labeled each line with the number of the equation that
   was used from the set of equations for substitution in Figure
   13.4. You should do that too.

   NOTICE that in carrying out these substitution derivations, we use
   the = operator. The idea is that the expression on the left of the
   = is *the same as* (equal to) the expression on the right. For
   instance, the expression indicated by (x + 5) [x â\206¦ 3] (the
   expression resulting from substituting 3 for x in x + 5 *is* the
   expression 3 + 5. The = relation differs from the â\207\223 ("evaluates
   to") relation. In particular, the former is symmetric; the latter
   is not.

   NOTICE that the result of substituting `3` for `x` in `x + 5` is
   `3 + 5` and not `8`. Do you understand why? *)

(*......................................................................
Exercise 3. Carry out the derivation for the semantics of the
expression `let x = 3 in x + 5`, using the semantics rules in Figure
13.5.
......................................................................*)

(* SOLUTION:

    let x = 3 in x + 5 â\207\223
                      | 3 â\207\223 3          (R_int)
                      | 3 + 5 â\207\223 8      (Exercises 2 and 1)
                      â\207\223 8              (R_let)

   Note the labeling of one of the steps with the prior results from
   previous exercises. *)

(* Now it's your turn. We recommend doing these exercises with
pencil on paper, rather than typing them in.

......................................................................
Exercise 4. Carry out the derivation for the semantics of the
expression `8 - 2`, using the semantics rules in Figure 13.5.
......................................................................*)

(*......................................................................
Exercise 5. Carry out the derivation for the semantics of the
expression `6 * 6`, using the semantics rules in Figure 13.5.
......................................................................*)

(*......................................................................
Exercise 6. What is the result of the following substitution according
to the definition in Figure 13.4?
```

```
    (x * x) [x â\206¦ 6]
......................................................................*)

(*......................................................................
Exercise 7. The set of 11 equations defining substitution in Figure
13.4 has an equation for function application. You'll need this
equation in some exercises below. Without looking at Figure 13.4,
what do you think such an equation should look like? Check your
understanding against Figure 13.4.
......................................................................*)

(*    (Q R)[x â\206¦ P] = ????    *)

(*......................................................................
Exercise 8. What is the result of the following substitution according
to the definition in Figure 13.4?

    ((fun x -> x * x) (x - 2)) [x â\206¦ 8]
......................................................................*)

(*......................................................................
Exercise 9. Carry out the derivation for the semantics of the
expression

    (fun x -> x * x) (8 - 2)

using the semantics rules in Figure 13.5.
......................................................................*)

(*......................................................................
Exercise 10. Finally, carry out the derivation for the semantics of
the expression

    let x = 3 + 5 in (fun x -> x * x) (x - 2)

using the semantics rules in Figure 13.5.
......................................................................*)

(*======================================================================
Part 2: Pen and paper exercises with the free variables and
substitution definitions

In this part, you'll get more practice using the definitions of FV and
substitution from the textbook (Figure 13.4). Feel free to jump ahead
to later problems if you "get it" and are finding the exercises
tedious. *)

(*......................................................................
Exercise 11: Use the definition of FV to derive the set of free
variables in the expressions below. Show all steps using pen and
paper. (You can see an example derivation for

       FV(fun y -> f (x + y))

in Section 13.3.2 of the textbook.)

1. let x = 3 in let y = x in f x y

2. let x = x in let y = x in f x y

3. let x = y in let y = x in f x y

4. let x = fun y -> x in x
......................................................................*)
```

```
(*......................................................................
Exercise 12: What expressions are specified by the following
substitutions? Show all the steps as per the definition of
substitution given in the textbook, Figure 13.4.

1. (x + 1)[x â\206¦ 50]

2. (x + 1)[y â\206¦ 50]

3. (x * x)[x â\206¦ 2]

4. (let x = y * y in x + x)[x â\206¦ 3]

5. (let x = y * y in x + x)[y â\206¦ 3]

......................................................................*)

(*......................................................................
Exercise 13: For each of the following expressions, derive its final
value using the evaluation rules in the textbook. Show all steps using
pen and paper, and label them with the name of the evaluation rule
used. Where an expression makes use of the evaluation of an earlier
expression, you don't need to rederive the earlier expression's value;
just use it directly.

1. 2 * 25

2. let x = 2 * 25 in x + 1

3. let x = 2 in x * x

4. let x = 51 in let x = 124 in x

......................................................................*)

(*======================================================================
Part 3: Implementing a simple arithmetic language.

You will now implement a simple language for evaluating `let` bindings
and arithmetic expressions. Recall the following syntax for such a
language from the textbook.

<binop> ::= + | - | * | /
<var>   ::= x | y | z | ...
<expr>  ::= <integer>
          | <var>
          | <expr1> <binop> <expr>
          | let <var> = <expr_def> in <expr_body>

......................................................................
Exercise 14: We've provided below type definitions that allow for
expressions implementing this syntax. Augment the type definitions to
allow for other binary operations (at least `Minus` and `Times`) and
for unary operations (at least Negate). Hint: Don't forget to extend
the type definition of `expr` to support unary operations as well.

When you're done, you should be able to specify expressions such as
the following:

    # Int 3 ;;
    - : expr = Int 3
    # Binop (Plus, Int 3, Var "x") ;;
    - : expr = Binop (Plus, Int 3, Var "x")
```

```
    # Unop (Negate, Int 3) ;;
    - : expr = Unop (Negate, Int 3)
    # Let ("x", Int 3, Binop (Plus, Int 3, Var "x")) ;;
    - : expr = Let ("x", Int 3, Binop (Plus, Int 3, Var "x"))
......................................................................*)

type varspec = string ;;

type binop =
  | Plus
  | Divide ;;

type unop =
  | NotYetImplemented ;;

type expr =
  | Int of int
  | Var of varspec
  | Binop of binop * expr * expr
  | Let of varspec * expr * expr ;;

(*......................................................................
Exercise 15: Write a function `free_vars : expr -> varspec Set.t` that
returns a set of `varspec`s corresponding to the free variables in the
expression.

The free variable rules in this simple language are a subset of those
found in Figure 13.4, but we encourage you to first try to determine
the rules on your own, consulting the textbook only as necessary.

To handle all of the set processing in the free variable rules --
unions and differences and so on -- we've provided a `VarSet` module
built using OCaml's `Set.Make` functor. (More documentation on the
`Set.Make` functor can be found at
<https://v2.ocaml.org/api/Set.Make.html>.)

You should get behavior such as this, in calculating the free
variables in the expression

    let x = x + y in z * 3      :

    # VarSet.elements
        (free_vars (Let ("x",
                         Binop (Plus, Var "x", Var "y"),
                         Binop (Times, Var "z", Int 3)))) ;;
    - : Lab9.VarSet.elt list = ["x"; "y"; "z"]
......................................................................*)

module VarSet = Set.Make (struct
                            type t = varspec
                            let compare = String.compare
                          end) ;;

let free_vars (exp : expr) =
  failwith "free_vars not implemented"

(*......................................................................
Exercise 16: Write a function `subst : expr -> varspec -> expr ->
expr` that performs substitution, that is, `subst p x q` returns the
expression that is the result of substituting `q` for the variable `x`
in the expression `p`.

The necessary substitution rules for this simple language are as
follows:
```

```
m[x â\206¦ P] = m                           (where m is some integer value)

x[x â\206¦ P] = P

y[x â\206¦ P] = y                    (where x and y are distinct variables)

(~- Q)[x â\206¦ P] = ~- Q[x â\206¦ P]        (and similarly for other unary ops)

(Q + R)[x â\206¦ P] = Q[x â\206¦ P] + R[x â\206¦ P]
                                  (and similarly for other binary ops)

(let x = Q in R)[x â\206¦ P] = let x = Q[x â\206¦ P] in R

(let y = Q in R)[x â\206¦ P] = let y = Q[x â\206¦ P] in R[x â\206¦ P]
                                (where x and y are distinct variables)

You should get the following behavior:

    # let example = Let ("x", Binop (Plus, Var "x", Var "y"),
                            Binop (Times, Var "z", Var "x")) ;;
    val example : Lab9.expr =
      Let ("x", Binop (Plus, Var "x", Var "y"), Binop (Times, Var "z", Var "x"))
    # subst example "x" (Int 42) ;;
    - : Lab9.expr =
    Let ("x", Binop (Plus, Int 42, Var "y"), Binop (Times, Var "z", Var "x"))
    # subst example "y" (Int 42) ;;
    - : Lab9.expr =
    Let ("x", Binop (Plus, Var "x", Int 42), Binop (Times, Var "z", Var "x"))
......................................................................*)

let subst (exp : expr) (var_name : varspec) (repl : expr) : expr =
  failwith "subst not implemented" ;;

(*......................................................................
Exercise 17: Complete the `eval` function below. Try to implement
these functions from scratch. If you get stuck, however, a good
(though incomplete) start can be found in section 13.4.2 of the
textbook.
......................................................................*)

(* Please use the provided exceptions as appropriate. *)
exception UnboundVariable of string ;;
exception IllFormed of string ;;

let eval (e : expr) : expr =
  failwith "eval not implemented"

(*......................................................................
Go ahead and test `eval` by evaluating some arithmetic expressions and
let bindings.

For instance, try the following expression, which is essentially

       let x = 6 in let y = 3 in x * y     .

# eval (Let ("x", Int 6,
                  Let ("y", Int 3,
                            Binop (Times, Var "x", Var "y")))) ;;
- : expr = Int 18

You now have a good start on the final project!
......................................................................*)
```