

(*
CS51 Lab 18
Environment Semantics

Objective:

This lab practices concepts of environment semantics. You'll carry out derivations using the various rule sets from Chapter 19, and gain intuition about dynamic versus lexical semantics and how stores work to allow mutability. The payoff exercises here are Exercises 9, 11, and 12.

Finally, you'll program a simple implementation of environments -- allowing lookup in and extension of environments -- which may be helpful with your work on the final project. *)

(*=====
Part 1: An environment semantics derivation

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

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

according to the semantic rules presented in Chapter 19, Figure 19.1, just as you did in Lab 9, Part 1 for substitution semantics.

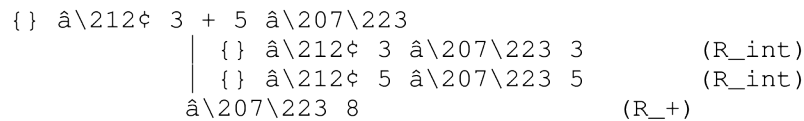
Before beginning, what should this expression evaluate to? Test out your prediction in the OCaml REPL. *)

(* 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.

.....
Exercise 1. Carry out the derivation for the semantics of the expression '3 + 5' in an empty environment.
.....*)

(* ANSWER:



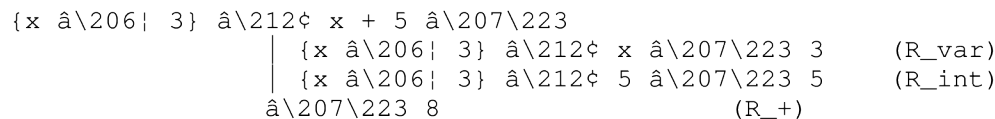
*)

(*.....
Exercise 2. Determine the result of evaluating the following expression 'x + 5' in the environment '{x -> 3}' by carrying out the derivation for

```
{x -> 3} -> x + 5 ???
```

.....*)

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



Again, we've labeled each line with the name of the equation that was used from the set of equations in Figure 19.1. You should do that too. *)

```
(*.....
Exercise 3. Carry out the derivation for the semantics of the
expression 'let x = 3 in x + 5' in an empty environment.
.....*)
```

(* ANSWER:

$$\begin{array}{l} \{\} \hat{\sphericalangle} \text{let } x = 3 \text{ in } x + 5 \hat{\sphericalangle} \\ \quad \left| \begin{array}{l} \{\} \hat{\sphericalangle} 3 \hat{\sphericalangle} \quad (R_int) \\ \{x \hat{\sphericalangle} 3\} \hat{\sphericalangle} x + 5 \hat{\sphericalangle} \quad (\text{Exercise 2}) \\ \hat{\sphericalangle} \quad (R_let) \end{array} \end{array}$$

Note the labeling of one of the steps with the result from a previous exercise.

The R_let rule specifies that the environment to be used in the third line in this derivation should be $E\{x \hat{\sphericalangle} v_D\}$, where

- * the metavariable E at this point is the empty environment $\{\}$,
- * the metavariable x is the object variable x
- * the metavariable v_D is 3.

Extending $\{\}$ with a mapping of x to 3 gives the environment $\{x \hat{\sphericalangle} 3\}$, which is exactly the environment that we use in line 3. The generation of the extended environment is carried out implicitly, the steps in doing so isn't spelled out explicitly here and needn't be in your own derivations.

*)

(* Now it's your turn. We recommend doing these exercises with pencil on paper. Alternatively, you might share a Google doc and work on developing the solutions there. After you've worked them out and verified them with staff, you can later copy them into your lab document. *)

```
(*.....
Exercise 4. Carry out the derivation for the semantics of the
expression 'x * x' in an environment mapping 'x' to '6', following
the rules in Figure 19.1.
.....*)
```

```
(*.....
Exercise 5. Carry out the derivation for the semantics of the
expression x - 2 in the environment mapping x to 8, following the
rules in Figure 19.1.
.....*)
```

```
(*.....
Exercise 6. Carry out the derivation for the semantics of the
expression (fun x -> x * x) (x - 2) in the environment mapping
x to 8, following the rules in Figure 19.1.
.....*)
```

```
(*.....
Exercise 7. Finally, carry out the derivation for the semantics of the
expression
```

$$\text{let } x = 3 + 5 \text{ in } (\text{fun } x \rightarrow x * x) (x - 2)$$

in the empty environment.

.....*)

(*=====

Part 2: Pen and paper exercises, dynamic vs. lexical semantics

The derivations you'll be constructing for these exercises can get *very complicated* -- as long as 33 lines for exercise 12 -- so you'll want to do them on paper, and check parts as you go. *)

(*.....

Exercise 8: For each of the following expressions, derive its final value using the evaluation rules in Figure 19.1. 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. Each expression should be evaluated in an initially empty environment.

1. $2 * 25$

2. $\text{let } x = 2 * 25 \text{ in } x + 1$

3. $\text{let } x = 2 \text{ in } x * x$

4. $\text{let } x = 51 \text{ in let } x = 124 \text{ in } x$

.....*)

(*.....

Exercise 9: Evaluate the following expression using the DYNAMIC environment semantic rules in Figure 19.1. Use an initially empty environment.

```
let x = 2 in
let f = fun y -> x + y in
let x = 8 in
f x
```

.....*)

(*.....

Exercise 10: For each of the following expressions, derive its final value using the LEXICAL evaluation rules in Figure 19.2. 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. Each expression should be evaluated in an initially empty environment.

1. $(\text{fun } y \rightarrow y + y) 10$

2. $\text{let } f = \text{fun } y \rightarrow y + y \text{ in } f 10$

3. $\text{let } x = 2 \text{ in let } f = \text{fun } y \rightarrow x + y \text{ in } f 8$

.....*)

(*.....

Exercise 11: Evaluate the following expression using the LEXICAL environment semantic rules in Figure 19.2. Use an initially empty environment.

```
let x = 2 in
let f = fun y -> x + y in
let x = 8 in
f x
```

.....*)

(*.....
 Exercise 12: For the following expression, derive its value using the LEXICAL evaluation rules for imperative programming in Figure 19.4. Show all steps using pencil and paper, and label them with the name of the evaluation rule used. The expression should be evaluated in an initially empty environment and an initially empty store.

```
let x = ref 42 in
(x := !x - 21; !x) + !x ;;
```

How does your result compare with the value of this expression as computed by the ocaml interpreter?

.....*)

(*=====

Part 3: Implementing environments

To represent an environment, we need to maintain a mapping from variable names to their values. For simplicity, we will consider only integer values here. In this part, you'll work on an especially simple implementation of environments. (Something like this may prove useful in implementing the final project!) A variable will be represented as a string, and an environment will be represented as an "association list" made of pairs of variables and their integer values.

The `List` module has some functions useful for manipulating association lists. You may want to make use of them here. See https://v2.ocaml.org/api/List.html#1_Associationlists.

*)

```
type varid = string ;;
type value = int ;;
type env = (varid * value) list ;;
```

(*.....

Exercise 13: Write a function `empty : unit -> env` that returns an empty environment.

.....*)

```
let empty () : env = failwith "empty not implemented" ;;
```

(*.....

Exercise 14: Write a function `extend : env -> varid -> value -> env` that extends an environment; that is, `extend e x v` should return an environment that maps variables to values just as `e` does *except* that it maps `x` to `v`. Make sure to handle the case where `x` is already in the environment.

.....*)

```
let extend (e : env) (x : varid) (v : value) : env =
  failwith "extend not implemented" ;;
```

(*.....

Exercise 15: Write a function `lookup : env -> varid -> value` that returns the value of a variable in the given environment, raising a `Not_found` exception if the variable has no value in the environment.

.....*)

```
let lookup (x : varid) (e : env) : value =
  failwith "lookup not implemented" ;;
```