1 Rackette

Your Rackette program will interpret racket programs at a high level. That is, it will unpack the syntax and semantics of a program and evaluate it, without needing to deal with the specific punctuation of racket code.

1.1 Environments

In Rackette, we will be using dictionaries to represent environments. You have already written a lookup procedure to look up values in a dictionary, which you should feel free to modify for use in this project. An environment in Rackette will be a list of tuples containing an ID of identifier and a VBuiltin.

1.2 Evaluation

To handle an expression, your eval function should take the expression and an environment as inputs and produce a value: for example, NumE 4 can be evaluated using just the following line:

| NumE n -> VNum n

To handle a definition, your eval function should take the definition and an environment as inputs and produce a new environment, which should be the input environment with a new binding added. Note that this does not modify any currently existing environment, because an environment is a list, and lists in OCaml are immutable.

1.3 The Top Level Environment

The initial top level environment of Racket should contain all the built in procedures and values for the names ‘true’ and ‘false’. Below is an example snippet from an initial top level environment:
initial_tle = [  
...  (ID "not", VBuiltin "<builtin:not>", not_func);  
...] ;;

let not_func: value list -> value = function  
  VBool hd :: [] -> VBool (not hd)  
  | hd :: [] -> failwith "not expects either true or false"  
  | _ -> failwith "not expects exactly 1 argument" ;;

2 Records

In OCaml, a record is a data structure that collects multiple values into a single structure. Example syntax for using a record:

> type point =  
{  
x : float ;  
y : float ;  
} ;;
> let my_point = {x = 5.; y = 3.} ;;  
> my_point.x +. my_point.y ;;  
- : float = 8.

Records share a lot of functionality with “and types” (tuples, etc.), but in some cases allow for much more readable programs. You might find them useful in Rackette, but are by no means required nor expected to use them in your project. However, they are an important concept in Homework 10. Although you can easily access the fields of a record using the syntax above, it may still sometimes be useful to pattern match with a record. The syntax for that is below:

match p with  
| {x = foo; y = bar} -> (*expression using foo and bar*)

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS 17 document by filling out the anonymous feedback form: http://cs.brown.edu/courses/csci0170/feedback