Lab 8: Environments
12:00 PM, Oct 29, 2017

Contents

1 Rackette Environment

Objectives

By the end of this lab, you will know:

- how to use environments for the Rackette project

By the end of this lab, you will be able to:

- make a new environment
- add a binding to an environment
- extend an environment by a collection of new bindings
- look up identifiers in an environment

1 Rackette Environment

Note: The code you write in this lab may end up in your Rackette project. Even though your lab partner might not be your project partner, this is fine!

In class we discussed environments (which are sequences of bindings) and how we could represent an environment as a list. We also talked about dictionaries (which are lists of key-value pairs), as well as the lookup function, which searches through the list for a pair containing the key you’re looking for, and then returns the value associated with the key. We can use a dictionary to represent an environment, so that lookup consists of finding a binding whose identifier is the one you’re looking up, and then returning the value for that binding.

In summary, we can say a binding is something like this, where the string component is the identifier:

```ocaml
type binding = string * value
```

It’s helpful, though, to define types that let Ocaml’s type-system help us. Strings will be used for multiple things in the Rackette project, and by defining several different types, each type a wrapper for a string that has a different purpose, we can avoid many tricky type bugs.

So for Rackette, rather than just treating an identifier as a string, we’re going to recommend this as the type-definition for the representation of an identifier:

```ocaml
type identifier = ID of string
```
And for values, we recommend naming a type as well. As was mentioned in class, for this assignment, we can choose a very simple version of “value” – something that allows for multiple possibilities, but isn’t yet complete:

```plaintext
type value = Int_val of int | String_val of string
```

Then a binding now becomes

```plaintext
type binding = identifier * value
```

Now that we’ve specified what a binding is, here are the things we need to do with an environment:

- Make a new empty environment (for creating the top-level environment, but also for creating the environment that’ll be part of every closure that we create)
- Add a binding to an environment
- Extend one environment by another. (When we extend $E$ by $E'$, the bindings in $E'$ should override those in $E$, so that if a name $s$ is bound in $E'$ and $E$, the value of $s$ in the extended environment is found from $E'$ rather than $E$.)
- Look up an identifier in the environment to find out the value it is bound to (or fail with in the event that the identifier is unbound). (An alternative design is to return a value option, returning None when the identifier doesn’t appear in the environment.)

**Task:** Write type definitions for bindings and environments (for which you’ll need the type-definition for values, but you can use a placeholder for that, like `type value = Int | String`, for instance).

Then write four procedures to: (1) create an empty environment, (2) add a binding to an environment, (3) extend one environment by another and return the resulting larger environment, and (4) look up the value associated with an identifier (represented as "ID of string") in an environment (or fail as described).

**Note:** Don’t overthink the four procedures.

Your code (aside from the check-expects) should be just a dozen lines or so. Since you’ll be using this code in the Rackette project, you have a strong incentive to get it right!

| You’ve reached a checkpoint! Please call over a lab TA to review your work.

That’s it for this week! Feel free to use the rest of this lab session to work on Eliza or ask questions about Rackette!

| Once a lab TA signs off on your work, you’ve finished the lab! Congratulations! Before you leave, make sure both partners have access to the code you’ve just written.

Please let us know if you find any mistakes, inconsistencies, or confusing language in this or any other CS17 document by filling out the anonymous feedback form: [http://cs.brown.edu/courses/cs017/feedback](http://cs.brown.edu/courses/cs017/feedback)