Lec 5: Closures

CS220: Programming Principles

Sang Kil Cha

Attendance Check

Note:

1. This slide appears at random time during the class.
2. This link is only valid for a few minutes.
3. We don't accept late responses.

Recap: Recursion

Another Example: Exponentiation

Compute the exponential of a given number.

Simple linear recursion.

```
let exp b n =
  if n = 0 then 1
  else b * exp (n - 1)
```

Tail-recursion.

```
let exp b n =
  let rec iter b counter product =
    if counter = 0 then product
    else iter b (counter - 1) (b * product)
  iter b n 1
```

Tail-recursion.

```
let exp b n =
  let rec iter b counter product =
    if counter = 0 then product
    else iter b (counter - 1) (b * product)
  iter b n 1
```

Can we make it faster?

Faster Algorithm

No need to multiply n times.

$$b^n = \begin{cases} (b^{n/2})^2 & \text{if } n \text{ is even.} \\ b \cdot b^{n-1} & \text{if } n \text{ is odd.} \end{cases}$$

Fast exp algorithm.

```
let isEven n = n % 2 = 0
let square n = n * n

let rec fastExp b n =
  if n = 0 then 1
  elif isEven n then square (fastExp b (n/2))
  else b * fastExp b (n - 1)
```

`elif` is equivalent to `else if`.

Measure Execution Time in REPL

```
#time
  exp 2 1000000

#time
  fastExp 2 1000000
```

Caveat: the result will be invalid due to integer overflow.

Scope

Locally Declared Identifiers

We learned from the previous lecture that let-bindings can be nested, but with a careful indentation.

```
let x = 1
let f x = x + x
f 10 // ?
let g a =
  let x = 10
  a + x
g 10 // ?
x // ?
```

Dynamic Environment

To understand the semantics of a program, we need to understand the environment in which the program is executed. The environment is a mapping from identifiers to values, and it changes through the execution of the program.

Example

```
(* A *) let x = 42
(* B *) let y = x + 1
(* C *) x + y
```

- At A, the environment is $\{\cdot\}$.
- At B, the environment is $\{\mathrm{x} \mapsto 42\}$.
- At C, the environment is $\{\mathrm{x} \mapsto 42, \mathrm{y} \mapsto 43\}$.

Is Initial Environment Empty?

Is Initial Environment Empty?

Although, it is ***not really empty***, we represent it as an empty set for simplicity.

Scope

The environment is effective only in a certain region of the program.

```
let myfunc x = // z is not in scope
  let y = x + 1
  y + y

let z = myfunc 10 // x is not in scope
```

Question

What's the value?

```
let x x =
  (let x = 10 in x + x) + x
x 10 // here?
```

Shadowing

Shadowing means that a binding in an inner scope hides a binding in an outer scope.
Shadowing does not affect the outer binding.

Question

What's the value?

```
let pi = 3.14
let area r = pi * r * r
let myarea =
  let pi = 6.0
  area 10.0 // here?
```

Let's assume that the body of a function is evaluated in the current dynamic environment (i.e., the environment at the time of the function call), what's the expected value?

What about F#?

What's the value of `myarea`? Why different?

An example function `area`.

```
let pi = 3.14
let area r = pi * r * r
let myarea =
  let pi = 6.0
  area 10.0 // ?
```

Static (Lexical) Scoping vs. Dynamic Scoping

Most programming languages use ***static scoping***, meaning that name resolution depends on the lexical context. In dynamic scoping, however, name resolution depends on the (dynamic) execution context.

Only a few languages support dynamic scoping, e.g., Emacs Lisp and LaTeX.

Why?

Static Scoping is Preferred

Because it is easier to understand and reason about. Programmers can easily determine the scope of a variable by looking at the source code.

How Do We Implement Static Scoping?

Each function declaration should remember the environment in which it is defined.

A ***closure*** is a data structure that stores a function body (the code) and the environment in which the function is defined.

Closure

We can evaluate functions into a value by means of a closure. A closure is a triple:

$$(\text{arg}, \text{body}, \text{env})$$

where arg is the argument expression, body is the function body expression, and the env is an environment.

Closure Example

An example function `area`.

```
let pi = 3.14
let area r = pi * r * r
let myarea =
  let pi = 6.0
  area 10.0 // ?
```

We can represent the closure of `area` as follows:

- `arg`: `r`
- `body`: `pi * r * r`
- `env`: $\{\texttt{pi} \mapsto 3.14\}$

Excercise

What's the value z?

```
let x = 42
let y = 24
let f x = x + y
let z =
  let y = 10
  f (x + y)
```

1. With lexical scoping?
2. With dynamic scoping?

Quiz #2

- The problem is publicly available at `https://github.com/KAIST-CS220/Quiz2`.
- This will be auto-graded (unlike the previous in-class activities).
- You can even see all the tests: `https://github.com/KAIST-CS220/Quiz2/blob/main/Tests/Tests.fs`.
- First, you should accept the assignment invitation.
- Then you wait for a minute or two until your own private repository is created.
- Finally, you can clone your own repository and start working on the quiz.

Quiz #2 (cont'd)

In this problem, you should write a function `collatz` that computes the number of steps required to reach 1, following the Collatz conjecture. The Collatz conjecture is a conjecture in mathematics that concerns a sequence defined as follows: start with any positive integer n. Then each term is obtained from the previous term as follows: if the previous term is even, the next term is one half of the previous term. If the previous term is odd, the next term is 3 times the previous term plus 1. The conjecture is that no matter what value of n, the sequence will always reach 1. More formally, the sequence can be represented as a function f as follows:

$$f(n) = \begin{cases} n/2 & \text{if } n \text{ is even} \\ 3n + 1 & \text{if } n \text{ is odd} \end{cases}$$

Question?