Implementing Lamda Functions

Based on …

- a blog by Matt Might
- "Closure conversion: How to compile lambda"

Outline

- Using closures to implement nested first-class functions in Python
- Closure terminology
- Using closures to implement lambda functions in Haskell

Nested First Class Functions in Python

What does the following code yield?

```
def f(x):
  def g():
    return x
  return g

a = f(42)
a()
```

Implementing Nested First Class Functions in C

What doesn't work

```
typedef int (*fp_t)(); // function pointer
int __global_x;

int g() {
  return __global_x;
}

fp_t f(int x) {
  __global_x = x;
  return g;
}
```

```
def f(x):
  def g():
    return x
  return g

a = f(42)
a()
```

```
# in Python what is this supposed to do?
a = f(10)
b = f(20)
```

Implementing Nested First Class Functions in C

What does work: closures

```
typedef int (*fp_t)(); // function pointer
typedef struct { int x; } G_ENV;
typedef struct { fp_t lambda; G_ENV env; } G_CLOSURE;

int g_lifted_lambda(G_ENV g_env) { return g_env.x; }

G_CLOSURE f_create_closure(int x) {
    G_ENV g_env;  g_env.x = x;
    G_CLOSURE clsr; clsr.env = g_env;
                    clsr.lambda = g_lifted_lambda;
    return clsr;
}

// a = f(10) in Python becomes . . .
a = f_create_closure(10)
// a() in Python becomes . . .
a.lambda ( a.env );
```

```
def f(x):
  def g():
    return x
  return g

a = f(10)
b = f(20)
a()
b()
```

Closures

Open lambda term

- Is a lambda function with parameters and some free variables.
- Example in Haskell: \x -> z
- z is a free variable, it's meaning is not fixed

Environment

- Is a mapping of variables to values.
- Example: M.fromList [("z",10)]

Closure

- "Is an open lambda paired with an environment that gives values to all of its free variables".
- Struct with a field for code and for the environment.

Another Example, now for Haskell

How do we convert the following Haskell code to C?

```
-- Haskell
foo x y = \p -> x + y + p
```

```
# Python code for reference
def foo(x, y):
  def anon(p):
    return x + y + p
  return anon
```