

Closures without Functional Programming

March 13, 2002

CS 131: Programming Languages

Callbacks/Upcalls

- When "system" code calls "user" code, this is called a **callback** or **upcall**.
 - Reverses the normal case of user code making calls to the underlying system.
 - Frequently the user function to call is passed to the system via a normal function call.
- Examples:
 - Having a function run every time a GUI button is pressed
 - Having a function run every time a packet arrives on the network
 - Other "event-driven" code

Callbacks in SML

- Suppose we have an interface something like:

```
type button
addCallback : button * (unit -> unit) -> unit
```

- Then, assuming we have buttons `button1` and `button2`:

```
fun f1 () = print "aha"
fun f2 () = print "oho"
val _ = addCallback(button1, f1)
val _ = addCallback(button2, f2)
```

Better Callbacks in SML

- This is ok since `f1` and `f2` are small, but what if each function had to open a new window just to display its string? Lots of code duplication...
- Better structure would be:

```
fun printer s = (fn () => print s)
val b1 = make_button (printer "aha")
val b2 = make_button (printer "oho")
```

Example: X-Windows

- The X Toolkit defines an event-driven model for X-windows
 - Can specify functions to be called when a particular event occurs (e.g. button pressed)
- The X Toolkit includes support for buttons
 - Can tell a button to invoke a function when pressed, by passing a function pointer to the `XtAddCallback` function
 - Function is specified via a function pointer

Callbacks in C

```
void f1(Widget w, XtPointer x, XtPointer y) {
    printf("aha");
}
void f2(Widget w, XtPointer x, XtPointer y) {
    printf("oho");
}
int main() {
    Widget button1, button2;
    /* initialization code for buttons here */
    XtAddCallback(button1, XmNactivateCallback, f1, 0);
    XtAddCallback(button2, XmNactivateCallback, f2, 0);
}
```

Constraints on C Function Pointers

- In C one can pass/return function pointers

```
void qsort
(void *base, size_t nel, size_t width,
 int (*compar)(const void *, const void *));
```

- But, any function passed to sort must be written completely by the programmer.
 - cannot "create" functions at run-time. (Why?)

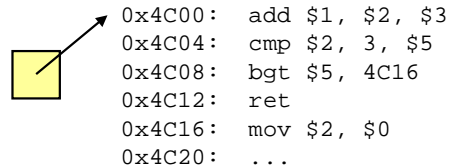
```
fun compare_nth_char n =
  fn (s1:string,s2:string) =>
    String.sub(x,n) > String.sub(y,n)
```

Better Callbacks?

- There is no direct equivalent in C to the printer higher-order function.
- If we want two buttons which do similar but slightly different things, must we write two completely separate functions?

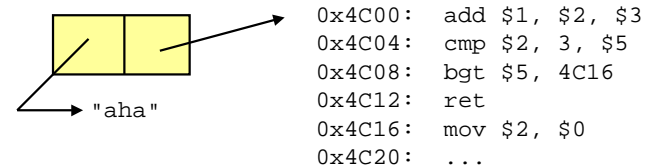
Closures vs. Code Pointers

- A function pointer in C is just the address of a piece of code *generated by the compiler*.



Closures vs. Code Pointers

- A function in SML is represented by a *closure*
 - Construct that contains both a pointer to code *and* some data that code can use



Manual Closures in C

```
void printer(Widget w, XtPointer client_data,
            XtPointer y) {
    printf("%s", client_data);
}

int main() {
    Widget button1, button2;
    /* ...initialization code for buttons here... */
    XtAddCallback(button1, XmNactivateCallback,
                  printer, "aha");
    XtAddCallback(button2, XmNactivateCallback,
                  printer, "oho");
}
```

Comparison: POSIX Threads

- The C call to create a POSIX thread is:

```
int pthread_create(pthread_t *new_thread_ID,
                  const pthread_attr_t *attr,
                  void * (*start_func)(void *),
                  void *arg);
```

"The thread is created executing `start_func` with `arg` as its sole argument. If the `start_func` returns, the effect is as if there was an implicit call to `pthread_exit()`"

Comparison: Java Callbacks

```
class AhaPrinter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.print("aha");
    }
}

class OhoPrinter implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.print("oho");
    }
}

JButton button1 = new JButton("B1");
JButton button2 = new JButton("B2");
button.addActionListener(new AhaPrinter());
button.addActionListener(new OhoPrinter());
```

Improved Java Callbacks

```
class Printer implements ActionListener {
    String s;
    Printer(String s) {this.s = s; }
    public void actionPerformed(ActionEvent e) {
        System.out.print(this.s);
    }
}

JButton button1 = new JButton("B1");
JButton button2 = new JButton("B2");
button.addActionListener(new Printer("aha"));
button.addActionListener(new Printer("oho"));
```

Closures vs. Objects

- What's the difference between an object and a closure?
- What's the difference between a method and a function?

Is SML Object-Oriented?

```
fun newcell() =
  let
    val r = ref 0
  in
    {contents = r,
     get      = (fn() => !r),
     set      = (fn n => (r:=n))}
  end

val mycell = newcell()
val _ = (#contents mycell) := 3
val x = (#set mycell) ((#get mycell)() * 2)
```