Lecture 25: Continuations & Data Abstraction

CSCI 131
Spring, 2011

Kim Bruce

So far ...

- Structured Programming
  - Goto considered harmful
- Exceptions
  - Structured jumps -- can carry a value
  - dynamic scoping of exception handler
- Continuations ...

Continuations

- Continuation of expression is remaining work to be done after evaluating expression
  - the future
  - Represented as a function, applied to value of exp
- Capture continuation
  - use it later to return to execution.
- Explicitly represented in Scheme, ML
- Have been important in compilers for functional languages, concurrency, web programming

Example 0: Roots of Quadratic

```
exception notQuadratic;
exception imaginaryRoots;
fun equal(x:real, y:real) = x <= y andalso x >= y;
fun roots(a, b, c) = let
   val discBase = (b*b) - (4.0*a*c)
   val denom = 2.0*a
 in
   if equal(denom, 0.0) then raise notQuadratic
   else if discBase < 0.0 then raise imaginaryRoots
   else let
         val disc = Math.sqrt(discBase)
        in
         if disc > 0.0 then [(-b + disc)/denom, (-b - disc)/denom]
         else [-b/denom]
        end
 end;
```

Example 1: Roots of Quadratic

```
fun checkQuad (x, n_continuation, e_continuation) =
 if equal(x, 0.0) then e_continuation() else n_continuation(x);

fun roots1(a, b, c) =
 let fun econt () = raise notQuadratic
     fun ncont (x) =
      let val discBase = (b*b) - (4.0*a*c)
      in
       if discBase < 0.0 then raise imaginaryRoots
       else let val disc = Math.sqrt(discBase)
            in
             if disc>0.0 then [(-b + disc)/x, (-b - disc)/x]
             else [-b/x]
            end
      end
 in
  checkQuad(2.0*a, ncont, econt)
 end;
```

Example 2: Roots of Quadratic

```
fun checkImaginary (x, n_continuation, e_continuation) =
 if x < 0.0 then e_continuation() else n_continuation(Math.sqrt(x));

fun roots2(a, b, c) =
 let
  fun econt () = raise notQuadratic
  fun ncont (x) =
   let
    fun econt () = raise imaginaryRoots
    fun ncont (disc) =
     if disc > 0.0 then [(-b + disc)/x, (-b - disc)/x]
     else [-b/x]
   in
    checkImaginary(b*b - 4.0*a*c, ncont, econt)
   end
 in
  checkQuad(2.0*a, ncont, econt)
 end;
```

Example 3: Roots of Quadratic

```
fun checkNumRoots(disc, continuation1, continuation2) =
 if disc > 0.0 then continuation1(disc) else continuation2();

fun roots3(a, b, c) =
 let fun econt () = raise notQuadratic
     fun ncont (x) =
      let fun econt () = raise imaginaryRoots
          fun ncont (disc) =
           let fun con1 (disc) = [(-b + disc)/x, (-b - disc)/x]
               fun con2 () = [-b/x]
           in
            checkNumRoots(disc, con1, con2)
           end
      in
       checkImaginary(b*b - 4.0*a*c, ncont, econt)
      end
 in
  checkQuad(2.0*a, ncont, econt)
 end;
```

Continuation-Passing Form

- Continuation-passing form of fcn f of n values is function w/one more -- continuation
- $f_{cpf}(x_1,\ldots,x_n,k) = k(f(x_1,\ldots,x_n))$
- So $f_{cpf}(x_1,\ldots,x_n,\lambda y.y) = \lambda y.y(f(x_1,\ldots,x_n)) = f(x_1,\ldots,x_n)$
- Use idea to derive cpf form

Using Continuations

- Used w/multiple threads w/separate stack
  - Blocked thread is represented as ptr to continuation
- CPS transform allows to rewrite programs so no need to ever return!
- Useful in web programming where no state.

Manipulating Evaluation Order

- What if actual params are expensive to evaluate, but aren't always used (and you are in an eager language)?
- Can suspend evaluation of e by replacing it by Delay(e) = fn () => e.
- Evaluate by Force(d) = d()
- Important to have as macros, not functions!

Call-by-need

- If have to evaluate several times then expensive.
- Cache answer once computed so can be accessed by other occurrences.

Implementing Call-by-need

```
datatype `a delay = Ev of `a |
                    UnEval of unit -> `a;

fun ev(Ev(x)) = x
  | ev(UnEval(f)) = f();

Delay(e) = ref( UnEval(fn () => e))

Force(d) = let
              val v = ev(!d)
           in
              (d := Ev(v);v)
           end;
```

Summary of Statements

- Progression from goto to higher-level abstractions:
  - Expression ⇒ function
  - Statement ⇒ procedure
  - control structure ⇒ iterator
- Modern control: iterators, exceptions, continuations, delay-force