Adding Recursion

February 20, 2002
CS 131: Programming Languages

Recursion

- The following code won't work as input to `eval`:

```
let fact be
   (N) => ((N==0) ? 1 : N * fact(n - 1)
in
    fact(10)
end
```

Recursive Function Values

- One fix (of many)
  - Let's just change the syntax for function values so that they can be refer to themselves!
- Abstract syntax: `rec` *`var`*`(`*`var`*`) =>` *`exp`*
  - Scope of both variables is just *exp*
- For example,

```
rec g(N) => (N==0 ? 1 : N*g(N-1))
```

- Not a definition, still just a value!
  - A value that is allowed to refer to *itself* using the name g

SML Comparison

- The SML equivalent to

```
rec g(N) => (N==0 ? 1 : N*g(N-1))
```

would be

```
let
   fun g(N) = if N=0 then 1 else N * g(N-1)
in
   g
end
```

- Code outside is not allowed to refer to the name g

Example Expressions

```
(rec f(x) => x+1) 3

(rec f(x) => f(x+1)) 3

let succ be (rec f(x) => x+1)
in let n be 4 in
   (succ n) + (succ (n+1))

let fib be
     rec g(x) =>  x == 0 ? 0 :
                  x == 1 ? 1 :
                    g(x-1) + g(x-2)
in
    fib(4)            (* not g(4) ! *)
```

Formal Semantics

- For simplicity, assume we start with the original substitution-based (eval) semantics
- Self-referential function values are still values and evaluate to themselves.
- Only interesting change is the application rule

Formal Semantics

$$\frac{exp_1 \Downarrow ((var) \Rightarrow \exp_3) \quad exp_2 \Downarrow value_2 \quad exp_3[var \leftarrow value_2] \Downarrow value_3}{\exp_1\ \exp_2 \Downarrow value_3}$$

$$\frac{exp_1 \Downarrow (\text{rec } var_1(var_2) \Rightarrow \exp_3) \quad exp_2 \Downarrow value_2 \quad exp_3[var_2 \leftarrow value_2][var_1 \leftarrow \text{rec } var_1(var_2) \Rightarrow \exp_3] \Downarrow value_3}{\exp_1\ \exp_2 \Downarrow value_3}$$

Example

- According to the semantics, to compute

```
(rec g(x) =>  x == 0 ? 0 :
                x == 1 ? 1 :
                  g(x-1) + g(x-2)) (3)
```

is a value applied to a value, it suffices to compute

```
3 == 0 ? 0 :
  3 == 1 ? 1 :
     ((rec g(x) =>  x == 0 ? 0 :
                      x == 1 ? 1 :
                        g(x-1) + g(x-2)))(3-1)
   +((rec g(x) =>  x == 0 ? 0 :
                      x == 1 ? 1 :
                        g(x-1) + g(x-2))) (3-2)
```