Closure Conversion

Handout #37

CS251 Lecture 29
April 19, 2005

Franklyn Turbak

Wellesley College

Translating Closures into FOFL

- Can translate FOBS to FOFL by lifting nested functions to top-level and adding extra arguments.
- Can we translate HOFL closures to FOFL? Yes – via a process called *closure conversion*.
- Closure conversion can be performed automatically, but it's often helpful to perform it manually to simulate higher order functions in languages like C and Java.

`sigma` Example in HOFL

```
(hofl (n) (test n)
  (def (sigma f lo hi)
    (if (> lo hi)
        0
        (+ (f lo) (sigma f (+ lo 1) hi))))
  (def (test w)
    (list (sigma sq 1 w)
          (sigma (scale 2) 1 w)
          (sigma (linear 3 4) 1 w)))
  (def (sq x) (* x x))
  (def (scale c) (fun (y) (* c y)))
  (def (linear a b) (fun (z) (+ (* a z) b))))
```

Closure-converting `sigma` in FOFL

```
(fofl (n) (test n)
  (def (sigma f lo hi)
    (if (> lo hi) 0
        (+ (applyClosure f lo) (sigma f (+ lo 1) hi))))
  (def (test w) (list (sigma (list (sym sq)) 1 w)
                      (sigma (scale 2) 1 w)
                      (sigma (linear 3 4) 1 w)))
  (def (sq x) (* x x))
  (def (scale c) (list (sym scale) c))
  (def (scaleHelper y c) (* c y))
  (def (linear a b) (list (sym linear) a b))
  (def (linearHelper z a b) (+ (* a z) b))
  (def (applyClosure clo arg)
    (bind name (nth 1 clo) ; Assume nth is 1-based list indexing
      (cond ((sym= name (sym sq)) (sq arg))
            ((sym= name (sym scale))
             (scaleHelper arg (nth 2 clo)))
            ((sym= name (sym linear))
             (linearHelper arg (nth 2 clo) (nth 3 clo)))
            (else (error "unknown closure"))))))
```

FOFL-PLUS

- Some languages with only top-level functions (particularly C) allow function values to be named, passed, returned, stored, but they *cannot* be created in any context (i.e., no closures).
- Model this in FOFL-PLUS = FOFL + two constructs:
  - (`fref` F) returns the function value denoted by F
  - (`fapp` E_{rator} E_{rand_1} $\ldots$ E_{rand_n}) invokes the function denoted by E_{rator} to the values denoted by the operands $E_{rand_1} \ldots E_{rand_n}$.

FOFL-PLUS Example

```
(fofl-plus (n) (list (app5 (fref inc))
                     (app5 (fref dbl))
                     (app5 (fref mul-n))))
  (def (inc x) (+ x 1))
  (def (dbl y) (* y 2))
  (def (mul-n z) (* z n))
  (def (app5 f) (fapp f 5)))
```

Closure-converting `sigma` in FOFL-PLUS

```
(fofl-plus (n) (test n)
  (def (sigma f lo hi)
    (if (> lo hi) 0
        (+ (applyClosure f lo)
           (sigma f (+ lo 1) hi))))
  (def (test w)
    (list (sigma (list (fref sq)) 1 w)
          (sigma (scale 2) 1 w)
          (sigma (linear 3 4) 1 w)))
  (def (sq x clo) (* x x))
  (def (scale c) (list (fref scaleHelper) c))
  (def (scaleHelper y clo) (* (nth 2 clo) y))
  (def (linear a b) (list (fref linearHelper) a b))
  (def (linearHelper z clo) (+ (* (nth 2 clo) z) (nth 3 clo)))
  (def (applyClosure clo arg) (fapp (nth 1 clo) arg clo))
```

Closure-converting `sigma` in JAVA, Part 1

```
interface IntFun {public int apply (int x);}

public static int sigma (int lo, int hi, IntFun f) {
  int sum = 0;
  for (int i = lo; i <= hi; i++) {
    sum = sum + f.apply(i);
  }
  return sum;
}

public static void main (String [] args) {
  int n = Integer.parseInt(args[0]);
  System.out.println(sigma(1, n, sqFun()));
  System.out.println(sigma(1, n, scaleFun(2)));
  System.out.println(sigma(1, n, linearFun(3,4)));
}
```

Closure-converting `sigma` in JAVA, Part 2

```
class Sigma {
  ⋮
  public static IntFun sqFun () {return new SqFun ();}
  public static IntFun scaleFun (int c) {return new ScaleFun (c);}
  public static IntFun linearFun (int a, int b)
    {return new LinearFun (a,b);}}

class SqFun implements IntFun {public int apply (int x) {return x * x;}}

class ScaleFun implements IntFun {
  private int c;
  public ScaleFun (int c) {this.c = c;}
  public int apply (int y) {return c * y;}}

class LinearFun implements IntFun {
  private int a,b;
  public LinearFun (int a, int b) {this.a = a; this.b = b;}
  public int apply (int z) {return (a * z) + b;}}
```

Closure-converting `sigma` in JAVA, Part 2'

Can instead use invocations of *anonymous inner classes*:

```
public static IntFun sqFun () {
  return new IntFun () {public int apply (int x) {return x * x;}};
}

public static IntFun scaleFun (final int c) {
  return new IntFun () {public int apply (int x) {return c * x;}};
}

public static IntFun linearFun (final int a, final int b) {
  return new IntFun () {public int apply (int x) {return (a * x) + b;}};
}
```