COM SCI 132 Week 8

Aidan Jan

May 20, 2024

How to Compile Lambda Expressions

In summary: we want to convert recursion to iteration.

Lambda Expressions → Tail Form → First-Order Form → Imperative Form

Form	**Item**	**Approach**
Tail	functions never return	continuations
First-order	functions are all top level	data structures
Imperative	functions take no arguments	register allocation

Recursion vs. Iteration

Recursion:

```
static Function<Integer, Integer> // in class Test
fact = n -> n == 0 ? 1 : n * Test.fact.apply(n - 1);
```

Iteration:

```
static int factIter(int n) {
    int a = 1;
    while (n != 0) { a = n * a; n--; }
    return a;
}
```

We want to convert the recursion to iteration.

0.1 Recursion to Tail Form

Recursion:

```
static Function<Integer, Integer> // in class Test
fact = n -> n == 0 ? 1 : n * Test.fact.apply(n - 1);
```

Tail Form: Uses continuation passing style (CPS).

```
static BiFunction<Integer, Function<Integer, Integer>, Integer>
factCPS = (n, k) -> n == 0 ? k.apply(1) : Test.factCPS.apply(n - 1, v -> k.apply(n * v));
```

To call a function that is written in CPS, use `.apply()`.

```
    factCPS.apply(4, v1 -> v1)
=   factCPS.apply(3, v2 -> (v1 -> v1)
                            .apply(4 * v2))
=   factCPS.apply(2, v3 -> (v2 -> (v1 -> v1)
                                  .apply(4 * v2))
                           .apply(3 * v3))
```

```
=   factCPS.apply(1, v4 -> (v3 -> (v2 -> (v1 -> v1)
                                         .apply(4 * v2))
                                  .apply(3 * v3))
                           .apply(2 * v4))
=   factCPS.apply(0, v5 -> (v4 -> (v3 -> (v2 -> (v1 -> v1)
                                                .apply(4 * v2))
                                         .apply(3 * v3))
                                  .apply(2 * v4))
                           .apply(1 * v5))
```

If we execute this. . .

```
factCPS.apply(0, v5 -> (v4 -> (v3 -> (v2 -> (v1 -> v1)
                                            .apply(4 * v2))
                                     .apply(3 * v3))
                              .apply(2 * v4))
                       .apply(1 * v5))
= factCPS.apply(1, (v4 -> (v3 -> (v2 -> (v1 -> v1)
                                              .apply(4 * v2))
                                       .apply(3 * v3))
                                .apply(2 * v4))
                          .apply(1))
= factCPS.apply(2, (v3 -> (v2 -> (v1 -> v1)
                                    .apply(4 * v2))
                             .apply(3 * v3))
                      .apply(2))
= factCPS.apply(6 -> (v2 -> (v1 -> v1)
                            .apply(4 * v2))
                     .apply(6))
= factCPS.apply(24 -> (v1 -> v1)
                      .apply(24))
= factCPS.apply(24)
= 24
```

Tail Form Grammar

The grammar is as follows:

$$\begin{aligned} TailForm ::= &\ \mathit{Simple} \\ \mid &\ \mathit{Simple}.\texttt{apply}(\mathit{Simple}_1, \ldots, \mathit{Simple}_n) \\ \mid &\ \mathit{Simple}\ \ ?TailForm\ :\ TailForm \end{aligned}$$

$$\begin{aligned} \mathit{Simple} ::= &\ Identifier \\ \mid &\ \mathit{Constant} \\ \mid &\ \mathit{Simple}\ \mathit{PrimitiveOperation}\ \mathit{Simple} \\ \mid &\ Identifier\ \texttt{->}\ TailForm \end{aligned}$$

- Evaluation of a Tail Form expression ($TailForm$) has **one** call which is the last operation.
- Evaluation of a Simple expression (Simple) has **no** calls.